CURSO DE

Microcontroladores PIC

SETEMBRO DE 2005

***

APRESENTAÇÃO

INTRODUÇÃO AO CURSO BÁSICO DE PIC

Nosso curso terá como base o PIC16F84-A, que atualmente ainda é o melhor PIC para iniciar os estudos dessa grande família de microcontroladores da Microcip. Como ele não tem todos os periféricos que a microchip pode oferecer, ele se torna simples de programar, e o melhor detalhe é que o set de instruções não muda quase nada de um PIC para outro, isto é, aprendendo o PIC16F84 você conseguirá aprender qualquer outro, bastando ler o datasheet par saber os nomes dos registros especiais... bom... mas isso fica para as próximas aulas.

Nosso Curso será muito mais prático do que teórico, o que implica que você terá que ter em mãos os recursos de um laboratório de eletrônica e um computador. Em termos de laboratório, o material é simples: Um protoboard para as montagens, um PIC16F84-A, um cristal de 4 MHz, capacitores, led´s chaves, display LCD, display de 7 segmentos, interface RS232, fonte de alimentação, conectores DB9, um gravador de PIC, etc... O computador acima ou igual a um Pentium 100 já é suficiente.

Durante as aulas vamos abordar os circuitos e os componentes que vão ser necessários você providenciar, a maioria dos componentes são de baixo custo.

Para finalizar, lembre-se que sua dedicação aos estudos e aos experimentos práticos é que vai dar subsídios para você realmente aprender a programar os microcontroladores.

Boa sorte

Ambiente Integrado de Desenvolvimento Mplab Versão 5.70

OBS. Existe versão mais nova do Mplab, mas essa é suficiente para o aprendizado básico.
querendo migrar para versão mais nova não há problema, o software é bem intuitivo...

Introdução

O MPLAB é um programa que tem a função de um gerenciador, para o desenvolvimento de projetos com a família PIC de microcontroladores. É distribuído gratuitamente pela Microchip, fabricante dos PIC's.
O MPLAB integra num único ambiente o editor de programa fonte, o compilador, o simulador e quando conectado às ferramentas da Microchip também integra o gravador do PIC, o emulador etc.
O Programa fonte, ou simplesmente fonte do programa é uma seqüência em texto, escrita numa linguagem de programação que será convertida em códigos de máquina para ser gravado no PIC.
O Compilador é o programa que converte o fonte em códigos de máquina.
O Simulador é programa que simula o funcionamento da cpu (PIC), conforme o programa fonte que está sendo desenvolvido.
O Projeto no MPLAB é um conjunto de arquivos e informações que diz ao ambiente integrado qual o PIC que estamos usando, qual freqüência de clock, qual a linguagem de programação usada, qual o layout das janelas etc. Enfim o projeto é o nosso trabalho de uma forma global. E para ele guardar todas essas informações basta salvar e fechar só o projeto, sem se preocupar em fechar todas as janelas abertas no ambiente integrado.
É importante lembrar que o MPLAB se integra ao ambiente Windows, permitindo cópia de arquivos, de textos de um aplicativo para outro de uma forma bem simplificada.

Adquirindo os programas necessários

Primeiro você tem que adquirir o software MPlab. A Microchip, que é o fabricante dos microcontroladores da família PIC, distribui gratuitamente este software, a intenção deles é que você compre os microcontroladores, por isso ela disponibiliza o software, bem como toda a sua documentação. O arquivo do software tem aproximadamente 13,4 Mb e o manual 2,7 Mb, os

arquivos estão disponíveis para download, porém se sua conexão não for de alta velocidade vai demorar algumas horas para baixar o arquivo.

Mplab(download 13,4 Mb)

Manual usuario Mplab.pdf (download 2,7 Mb) *

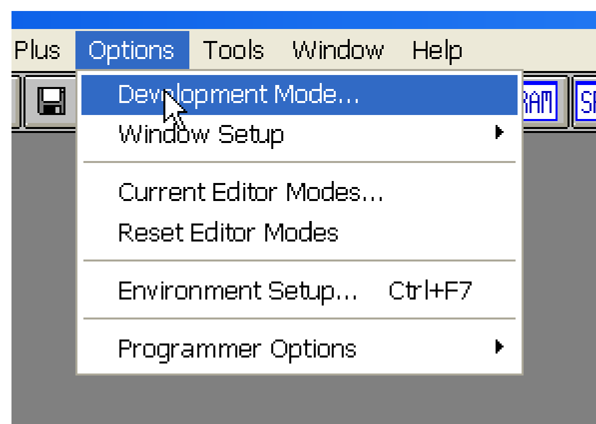
Configurando o MPlab para usar o PIC16F84-A com um cristal oscilador de 4 Mhz

Em nosso Curso, vamos montar um circuito básico para os experimentos práticos, onde usaremos um microcontrolador PIC16F84-A com um cristal oscilador de 4 Mhz. Como o MPlab serve para qualquer microcontrolador da linha PIC, é necessário configurá-lo para cada tipo de pic que vamos trabalhar num projeto. no nosso caso basta fazer esse procedimento uma única vez, pois o software guarda a última informação de configuração, e como só vamos usar o 16F84 não há necessidade de ficar alterando.

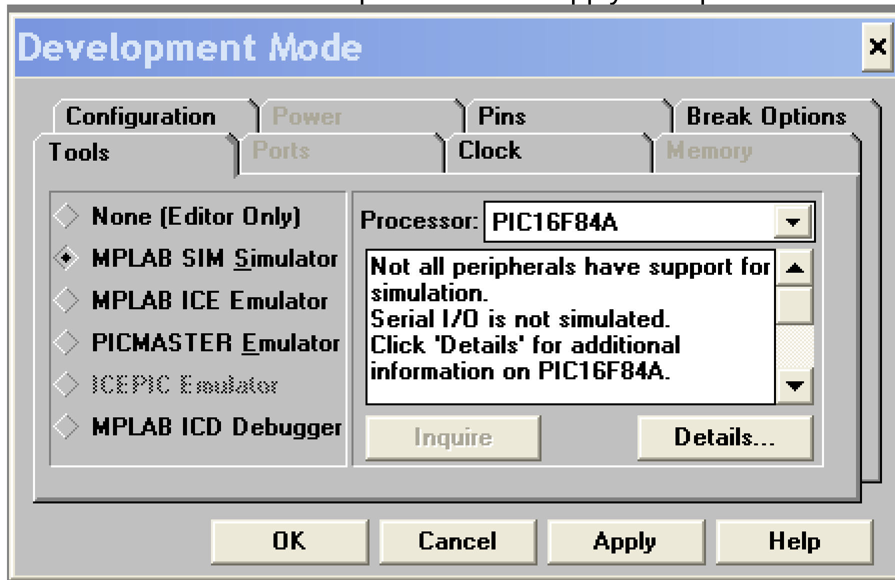
Definindo o PIC e o modo Simulador

Podemos configurar o MPLAB para ser apenas um editor de projetos, (none Editor Only), ou configurar como Editor mais Simulador, (MPLAB-SIM Simulator), no nosso caso vamos habilitar como simulador:

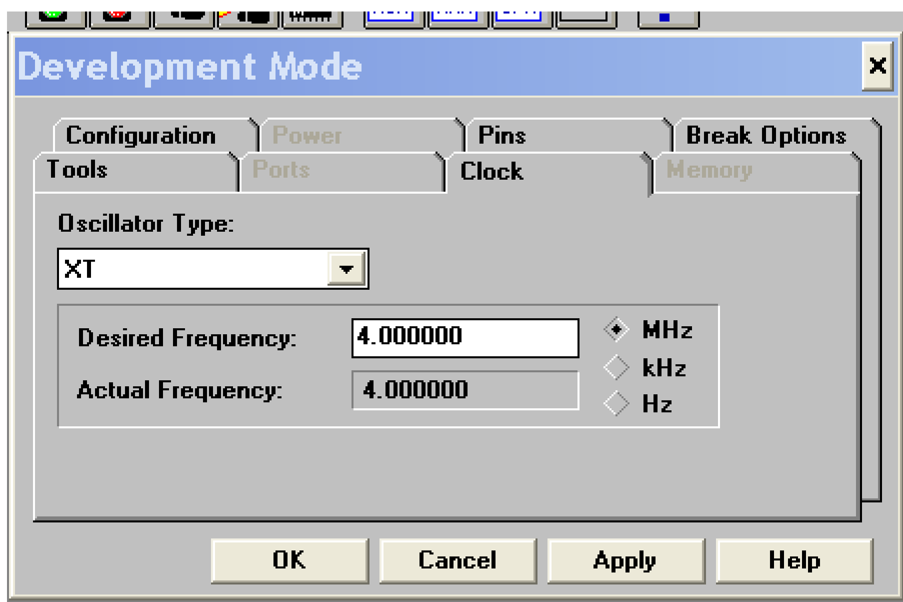
Selecione Options>Development Mode no Menu



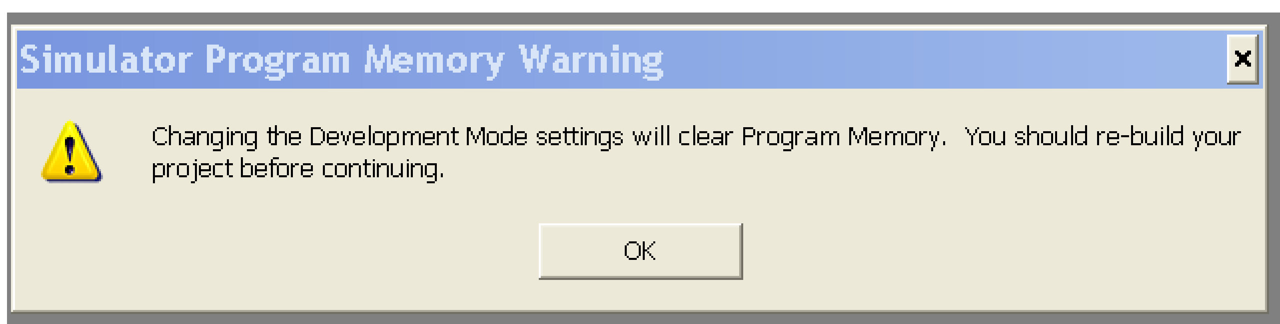
Click em tools para selecionar o Simulador e o tipo do PIC para o projeto. Click em MPLAB-SIM e escolha o PIC16F84-A depois click em Apply e depois em em Clock para configurar o cristal.



Escolha o Oscilator Type como XT e o Desired Frequency em 4.000000 MHz, esses 6 zeros depois do ponto é a presisão que dispomos para escolher a frequência, mas na prática nós dizemos 4 MHz. Agora Clique em OK



Se aparecer alguma mensagem do tipo que você vê abaixo, Responda OK.



Pronto seu Ambiente integrado de desenvolvimento está configurado

Organizando seus projetos

É aconselhável que cada projeto esteja numa pasta, (diretório), própria, isso é conveniente, pois o MPLAB gera uma série de arquivos para cada projeto e estando em pastas separadas fica fácil você fazer uma cópia do projeto, com essa cópia você pode desenvolver seu projeto em computadores diferentes sem perder as configurações e informações.

Passo 01
Criar um diretório na raiz do seu HD

Crie um diretório na raiz de seu HD onde você vai colocar todas as pastas de projetos do MPLAB, isso facilita a organização de seus arquivos:
Exemplo: crie em C: uma nova pasta com o nome _pic o traço baixo, ou under line antes da palavra pic, serve para que essa pasta seja exibida sempre no início do windows explorer, mas você pode escolher qualquer nome. Feche ou minimize o Mplab que está aberto para facilitar o visual.



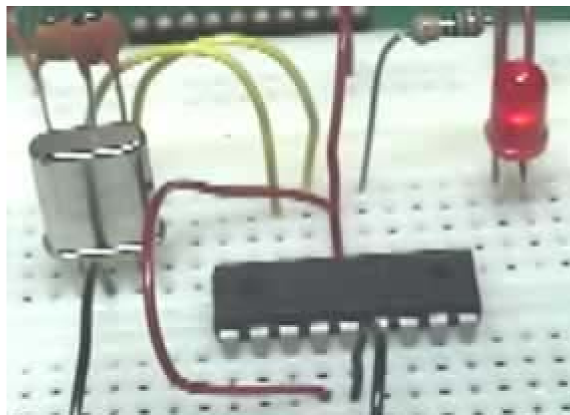
Passo 02
Criar uma pasta diferente para cada projeto dentro desse diretório _pic que acabamos de criar

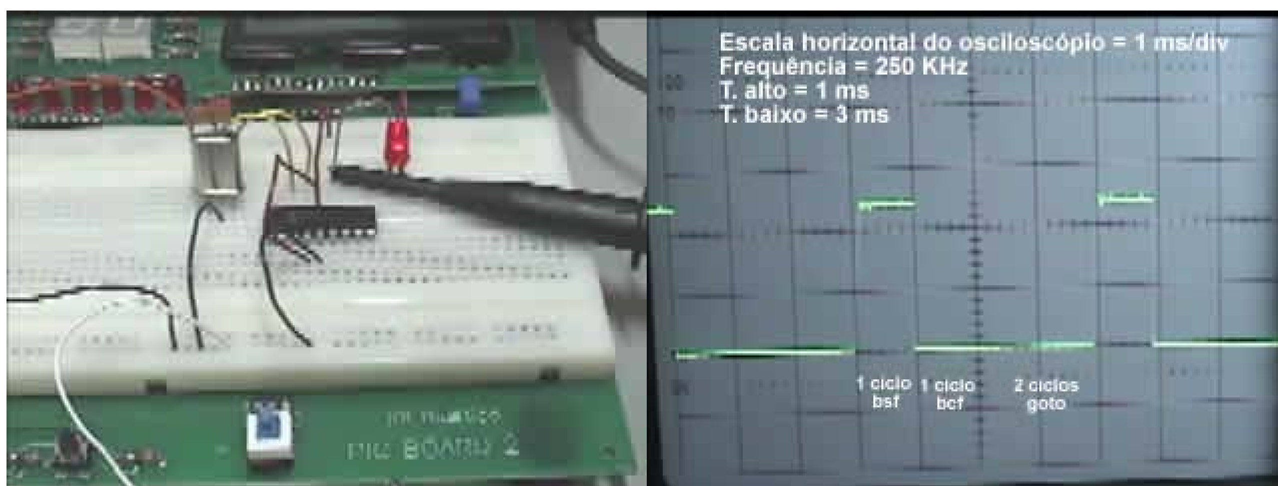
Entre no diretório C:>_pic ou outro que você criou: e crie uma nova pasta com o nome do seu novo projeto. Escolha um nome que seja fácil de lembrar o que é o projeto. Por exemplo:
Crie uma pasta com o nome astavel, nesta pasta iremos criar o nosso primeiro projeto, que vai fazer um pino do PIC gerar uma forma de onda quadrada, oscilador astável. Procure sempre escrever nomes com 8 caracteres, isso ainda é uma deficiência do MPLAB que traz alguns resquícios do bom e velho dos.



Pronto, agora já temos uma estrutura de diretorios para os nossos projetos, toda vez que falamos de um projeto, estamos nos referindo ao conjunto de arquivos que o MPlab vai gerar a partir de um código fonte que iremos criar para atender um circuito eletrônico, então é muito comum nossos projetos se chamarem: alarme, controle, pulsos, piscaled, etc. como também pode chamar: cliente1, cliente2...

Primeiro Projeto
Um gerador de onda quadrada (*)

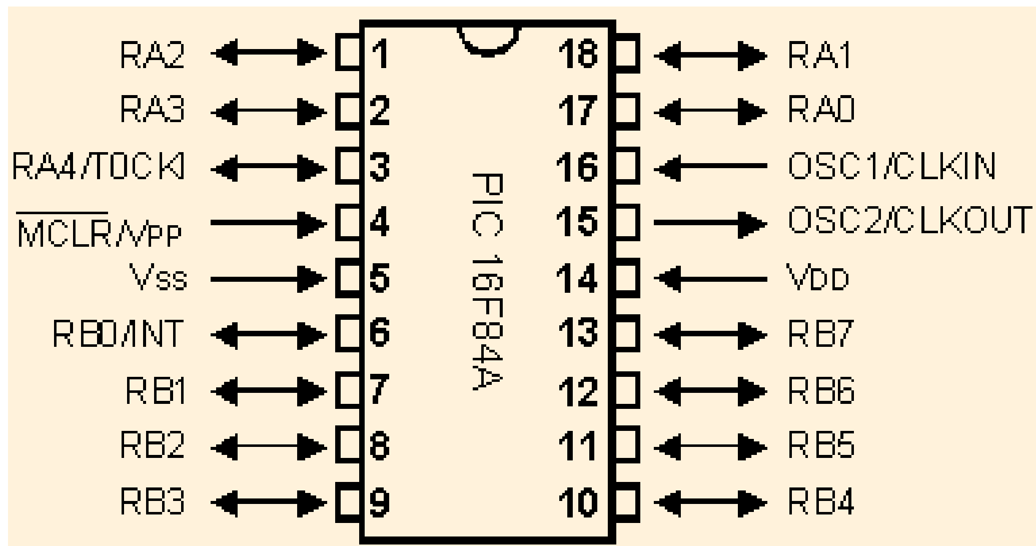




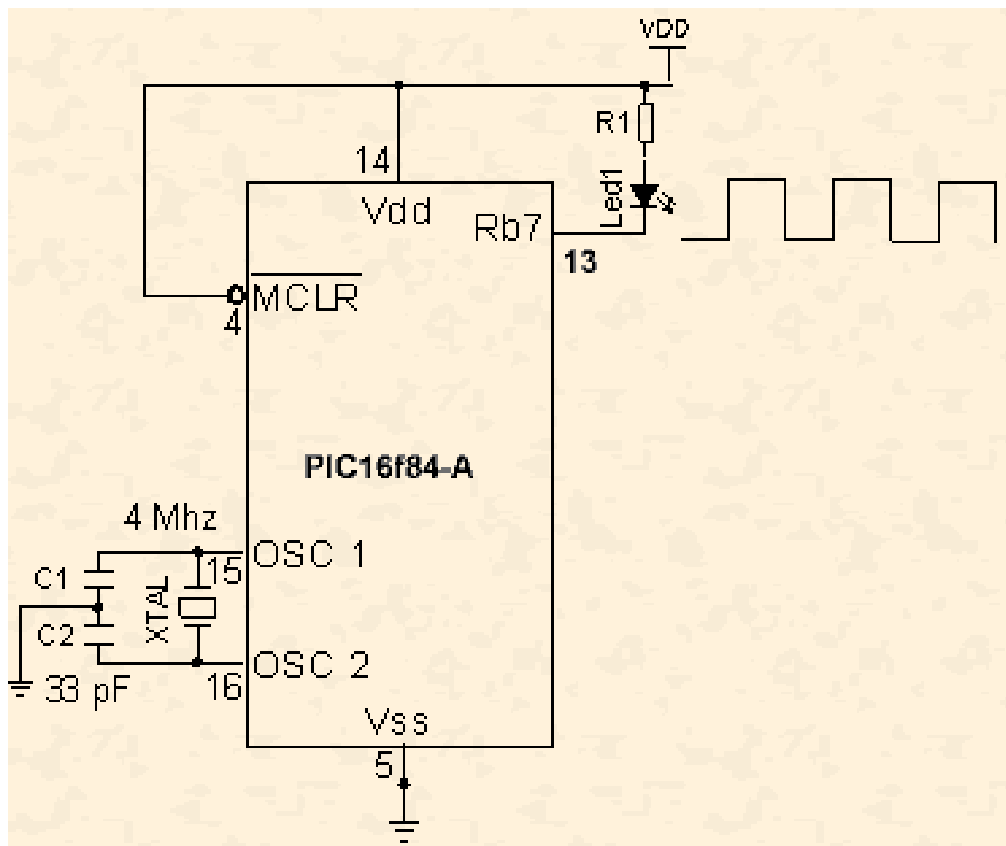
(*) Na verdade é uma onda retangular, com tempo alto de 1 milisegundo e tempo baixo de 3 milisengudos, o led no circuito apaga e acende tão rápido, que nossa vista vê apenas aceso, mas com um osciloscópio é possível monitorar a saída.

OBS: O PIC16F84 só difere do PIC16F84A na velocidade máxima do cristal oscilador externo, portanto todo o nosso curso pode ser feito com qualquer um deles, pois usaremos nos exemplos apenas cristal de 4 MHz.

Para nosso primeiro projeto, vamos fazer um programa para o PIC 16F84-A, de tal forma que o pino que tem o nome RB7(pino 13) vai pulsar de forma astável, ou seja, gerando uma onda retangular, como mostrado na tela do osciloscópio acima. Não falamos ainda sobre o PIC, mas ainda não há necessidade, quero que primeiro você domine a ferramenta de desenvolvimento, então não se preocupe com o PIC propriamente dito, tudo vem a seu tempo. Mas só pra matar a curiosidade a microchip dá nomes aos pinos dos controladores, que logicamente referem-se `as suas funções. Veja a seguir:



Nosso projeto vai ser usado no seguinte circuito:



Fique tranquilo, nas aulas posteriores iremos estudar mais profundamente o microcontrolador.

Bom vamos lá: este circuito sozinho, não faz nada... os microcontroladores da microchip, quando limpos, as configurações internas vem por padrão com os pinos todos em alta impedância, então realmente no circuitinho acima nada vai acontecer. Então para fazer ele gerar uma onda quadrada, temos que escrever um programa, e depois gravá-lo no PIC.

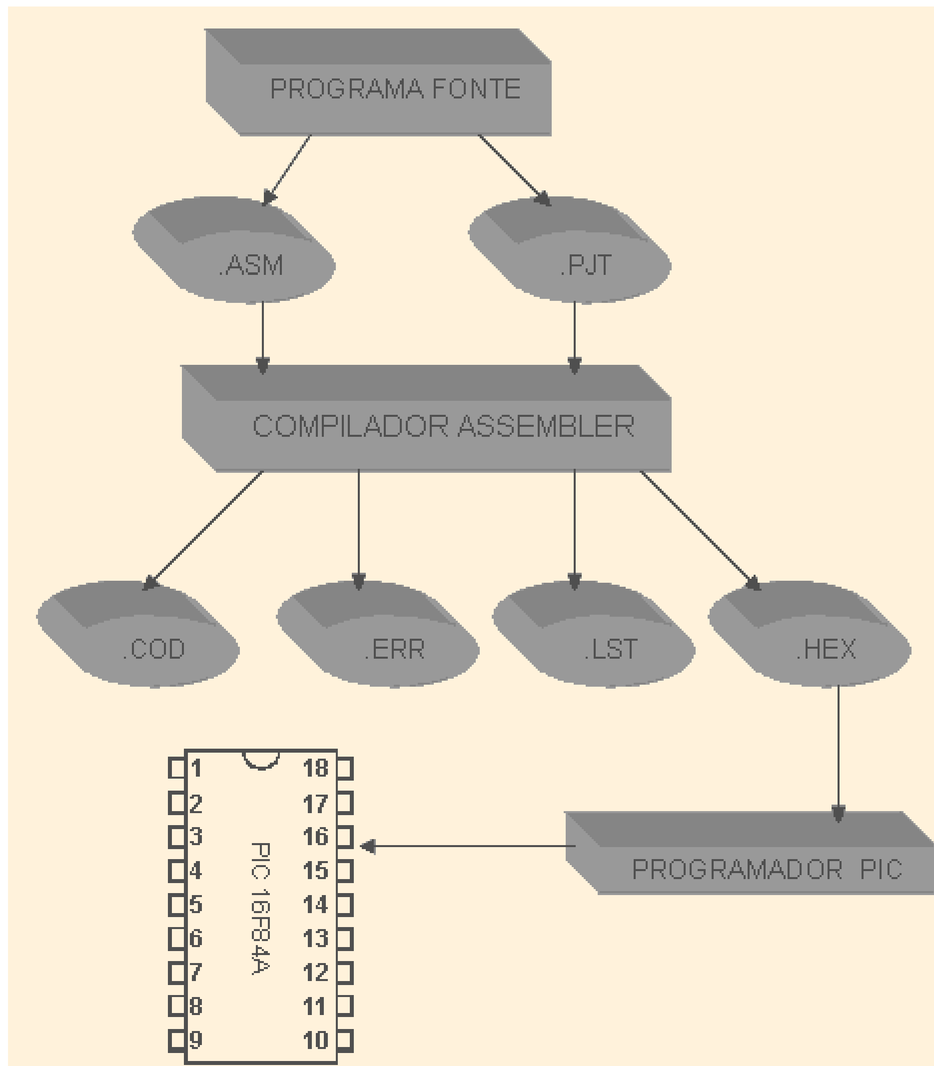
Todo sistema microprocessado ou microcontrolado necessita de um programa (software) para comandá-lo. O microcontrolador irá obedecer fielmente todas as ordens que forem atribuídas.

Um programa é constituído por um conjunto de instruções em seqüência, onde cada uma identificará precisamente a função básica que o PIC irá executar. Cada instrução é representada

por um código de operação (OPCODE - do inglês, Operation Code) de 14 bits, tamanho exato de cada locação de memória de programa, no caso do PIC 16F84.

O programa será escrito através de instruções mnemônicas (o PIC 16F84-A possui 35), podendo ser utilizado um editor de texto comum, ou como no nosso caso, o ambiente de desenvolvimento Mplab. Logo após a edição do programa fonte, será feita a compilação (transformar a linguagem de texto em linguagem apropriada para que o PIC possa entender) e finalmente gravar o PIC. A figura abaixo mostra o fluxograma das operações necessárias até a gravação do PIC (utilizando o MPLAB).

O MPlab vai gerenciar todos esses arquivos e os programas necessários para simulação, compilação etc. num único ambiente.



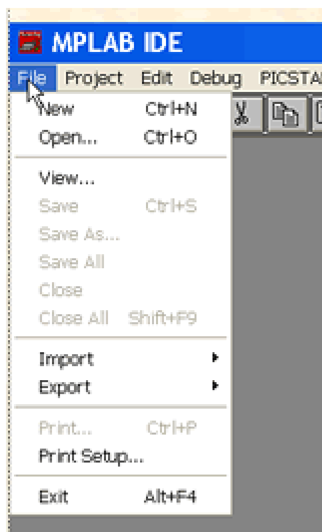
O Programa fonte é um arquivo de texto com as instruções e sua extensão é asm, que é o mnemônico de ASseMbler

O Projeto é gerenciado por informações gravadas num arquivo com extensão pjt (ProJeTo), depois de compilado o MPlab gera 4 arquivos importantes no projeto: o .COD o .ERR o .LST e o .HEX, depois veremos cada um desses arquivos, o que são e pra que serve.

Vamos começar:

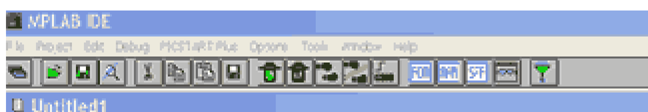
PASSO 01

Abra o MPlab Clique em File e depois em New



PASSO 02

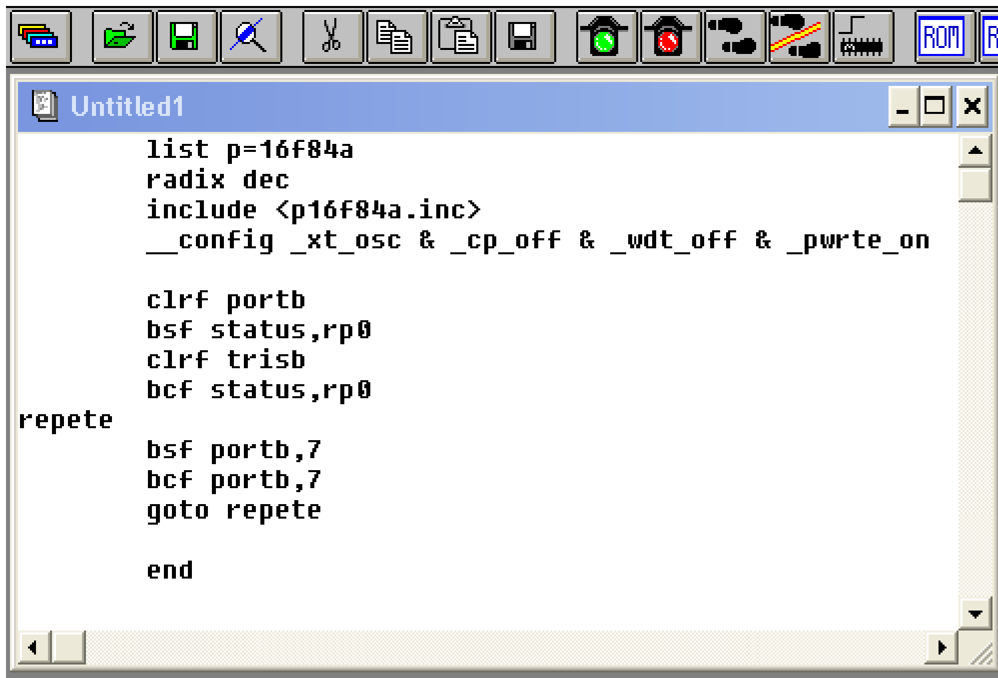
Isso vai abrir o editor de texto do ambiente de desenvolvimento, nesse momento abre-se uma caixa de diálogo, avisando que você não tem nenhum projeto aberto, e pergunta se quer fazer um novo. Responda Não (NO), depois que editarmos o fonte é que iremos fazer o projeto.





PASSO 03

Agora vamos digitar o Fonte: Com a tecla TAB ou dando espaços (8), comece sempre as instruções do código fonte numa determinada coluna, no MPlab basta pular pelo menos uma que o compilador já entende como uma instrução. Bom, como falamos em "compilador" já é hora de saber o que é isso. Compilador é um programa interno ao MPlab que transforma o código fonte de texto (.asm), para o código hexadecimal (.hex), que é o arquivo que vai ser enviado serialmente ao PIC (gravado). Para isso tudo que escrevemos no código fonte é para o compilador ler, interpretar e codificar para hexadecimal. Todo texto que ele ler após a primeira coluna ele entende como instrução. no nosso primeiro código fonte tem uma única linha que está na coluna zero, nesse caso o compilador entende como um rótulo, chamamos de label, que serve para marcar uma posição no código fonte, e de forma amigável podemos facilmente nomear as rotinas que se seguem logo após. então com muita atenção digite o texto como na figura abaixo:



Se preferir selecione o texto abaixo copie e cole no editor do MPlab acertando as tabulações.

```
list p=16f84a
radix dec
include <p16f84a.inc>
__config _xt_osc & _cp_off & _wdt_off & _pwrte_on

clrf portb
bsf status,rp0
clrf trisb
bcf status,rp0
repete
bsf portb,7
bcf portb,7
goto repete

end
```

Explicação do código fonte da aula 04:

```
list p=16f84a
radix dec
include <p16f84a.inc>
__config _xt_osc & _cp_off & _wdt_off & _pwrte_on

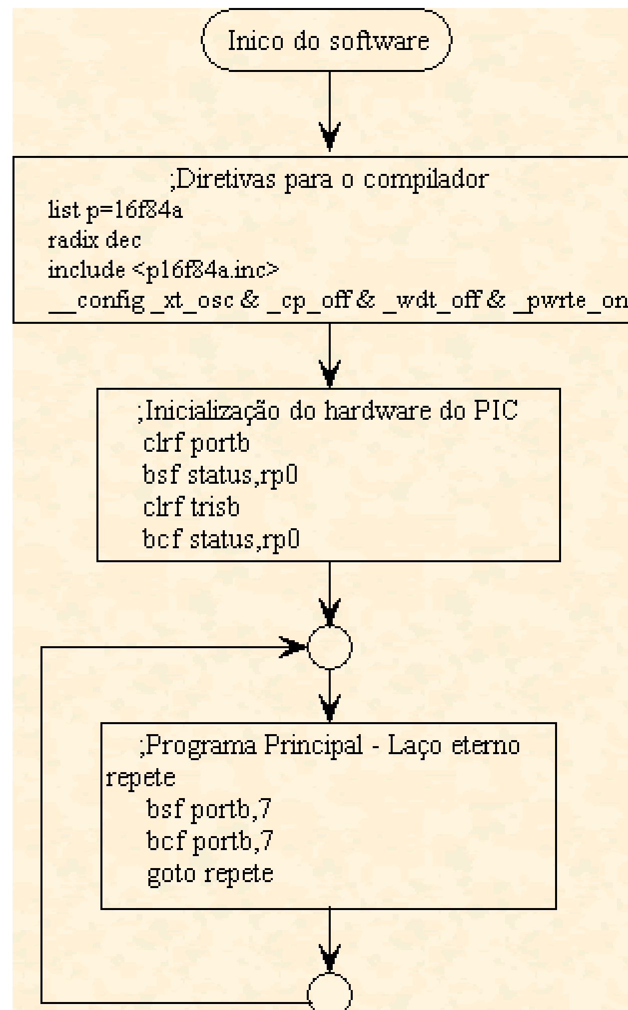
clrf portb
bsf status,rp0
clrf trisb
bcf status,rp0

repete

bsf portb,7
bcf portb,7
goto repete

end
```

Fluxograma do software



No fluxograma, vemos 3 etapas essenciais na programação de um PIC, que são: 1)As diretivas para o compilador 2)A inicialização do hardware do PIC 3)O programa principal.

1. As diretivas vão informar ao compilador qual o PIC que estamos desenvolvendo o software, que tipo de numeração estamos usando no nosso código fonte, quais os arquivos que vão ser anexados etc.
2. Na inicialização do hardware do PIC vamos programar quais os periféricos internos que vamos utilizar, qual pino vai ser saída ou entrada, situação inicial do seu circuito, etc.
3. O Programa principal, é como o próprio nome diz, o principal, aquilo que estamos querendo que o circuito execute. O detalhe importante é que sempre temos de fazê-lo em laço eterno, ou seja, deve eternamente executar esta rotina.

Vamos agora ver o que significa instrução por instrução

list p=16f84a

Esta instrução diz ao compilador que ele deve fazer a conversão do código fonte de texto para o padrão do PIC16F84-A

radix dec

Esta diz ao compilador que todo número que aparecer no texto, e não tiver nenhuma "marca" será entendido como dec=decimal.

include <p16f84a.inc>

Esta diz para o compilador incluir no nosso código fonte, um arquivo da microchip chamado p16f84a.inc, que está no diretório onde o Mplab está instalado, (C:\arquivo de programas\Mplab), onde está escrito as equivalências dos nomes dos registros e os respectivos endereços físicos, possibilitando nomes e abreviações mais amigáveis do que ficar guardando números.

__config _xt_osc & _cp_off & _wdt_off & _pwrte_on

Esta instrução configura o hardware interno do PIC, no nosso caso (__config, "dois traço baixo, ou underline, + config)é a instrução configurar, as strings, grupo de letras, entre os &, (detalhe: barra de espaço+&+barra de espaço), O _xt_osc, (agora só um traço baixo), significa configurar para um oscilador com cristal(vamos usar um de 4MHz); _cp_off é desabilitar o código de proteção, isso significa que após gravarmos o PIC é possível lê-lo; O _wdt_off é desabilitar o "watch dog", depois falaremos sobre isso; 0 _pwrte_on é para habilitar o "power on reset", internamente, quando ligamos a alimentação no PIC, ele fica em reset por 72 ms, tempo suficiente para estabilização do circuito.

clrf portb

Esta instrução é o "CLeaR File" clrf é o mnemônico coloca zeros num registro inteiro, no pic os registros são de 8 bits, isto é um byte, a Microchip chama esses registros de file, então as instruções do pic que se refere a registros de memória leva sempre a letra f, (portb) é o argumento dessa instrução, é nome do file que a instrução vai "encher de zeros", na verdade esse argumento tinha que ser um número, o número do endereço da memória ou registro, mas com aquele arquivo do "include", 16f84a.inc, podemos escrever portb, que isso equivale a 06, depois vamos ver isso... O menemônico portb se refere então ao registro de memória que controla uma porta de entrada e saída, a PORTa B, do PIC 16F84-A

bsf status,rp0

Esta instrução é o "Bit Set File" bsf faz com que um único bit pertencente a um registro seja alterado para 1. o status e o rp0 são argumentos da instrução, a instrução tem a seguinte sintaxe: bsf f,b onde o f é o endereço do registro (file) e b é o número do bit a ser alterado, como podemos trabalhar com mnemônico, não precisamos decorar números, então status é um registro especial no pic em que cada bit está relacionado com uma parte do hardware do microcontrolador, depois vamos estudar esses registros; O rp0 é o nome de um bit dentro do status, que altera o banco de memórias de dados que o PIC vai endereçar, no pic16f84 temos 2 bancos e o RPO=0 endereça o banco 0, e quando RP0=1 endereça para o banco 1, depois explicaremos melhor o que é isso.

clrf trisb

Esta instrução é o "CLeaR File" clrf faz com que todos os bits de um file vão pra zero, o trisb é o argumento da instrução, no caso o nome de um file ou registro especial do pic, que controla todos os pinos da porta B, ou portb. Então esta instrução vai garantir que no início do programa todos os pinos da porta B que estiver como saída vão pra nível lógico zero.

bcf status,rp0

Esta instrução é o "Bit Clear File" bsf faz com que um único bit pertencente a um registro seja alterado para Zero. O status e o rp0 são argumentos da instrução, a instrução tem a seguinte sintaxe: bsf f,b onde o f é o endereço do registro (file) e b é o número do bit a ser alterado, verifique que é o inverso de bsf. Com esta instrução voltamos para o banco Zero de dados.

repete

O repete é um label, ou rótulo, é uma sequência de string´s que marca uma posição no programa, poderia ser qualquer sequencia de caracteres alfanuméricos, aceita também o under line(_), e no máximo 32. Normalmente colocamos nomes sugestivos, que facilitam relacionar o que faz aquela sequência de instruções que vem logo após o label.

bsf portb,7

Já vimos a instrução bsf, aqui ela manda o bit 7 do file portb ir para nível lógico 1, no nosso circuito esse bit 7 do file portb é o RB7, que está ligado no led, analisando o circuito elétrico, esse nível 1, leva o pino (13) à 5V o que vai fazer APAGAR O LED.

bcf portb,7

Já vimos esta instrução também, o bcf, aqui manda o bit 7 do file portb ir para nível lógico zero, analisando o circuito elétrico, esse nível zero, leva o pino (13) à 0V o que vai fazer ACENDER O LED.

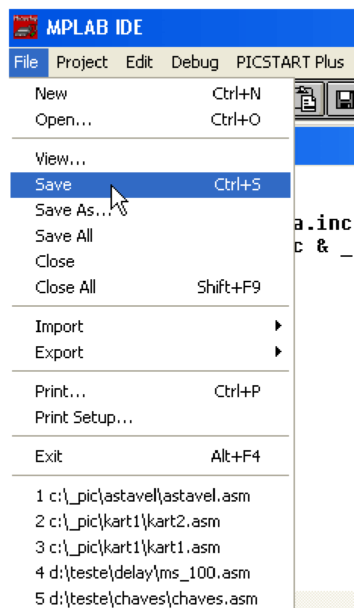
goto repete

Esta instrução é o " GO TO", vai para, o repete é o argumento, então a instrução (goto repete) manda o pic ir para a posição de programa que batizamos de repete, isto no circuito é fazer o led apagar e acender de novo infinitamente.

end

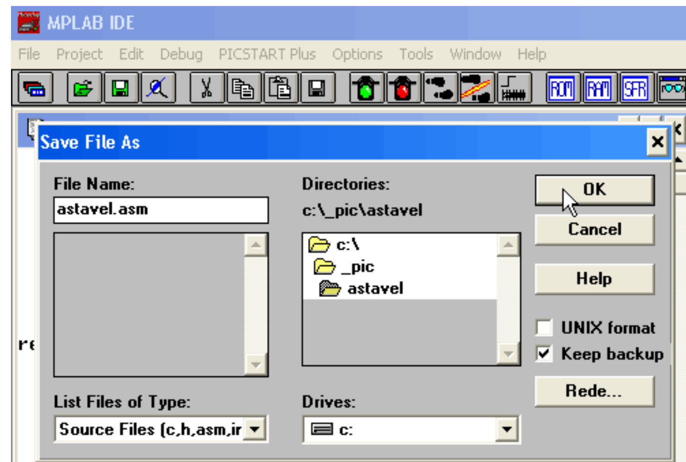
Esta é uma informação ao compilador dizendo que o código fonte terminou. Não é instrução do PIC.

PASSO 04 Depois do fonte digitado, temos que salvá-lo com extensão .asm naquela pasta astavel que criamos para o projeto. para isso clique em file > save

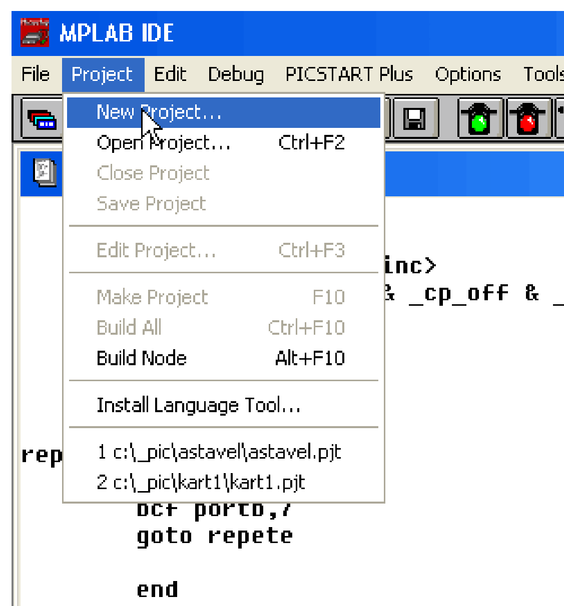


Passo 4.1

Procure a pasta astavel que criamos para o projeto, e digite no File name astavel.asm e clique em OK

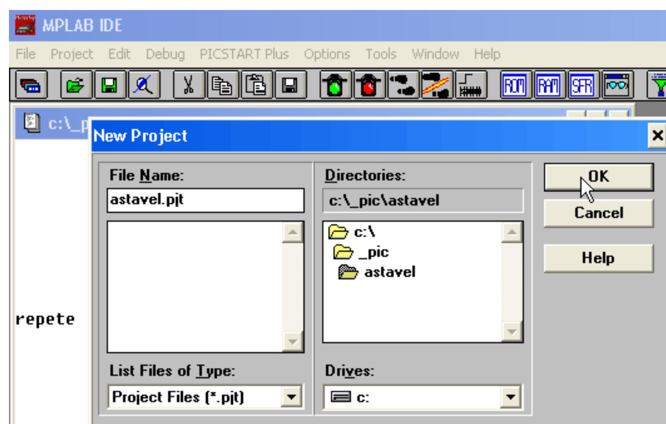


Passo 4.2
Clique em Project > New Project



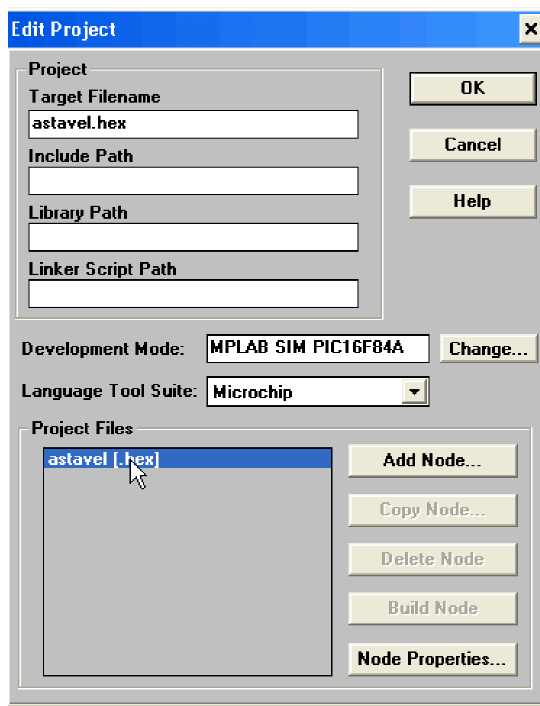
Passo 4.3

Procure a pasta astavel e digite no File Name astavel.pjt e clique em OK. Nesse caso você pode digitar só astavel que a extensão o próprio MPlab completa. Mas no nome do fonte você tem que colocar a extensão. Tome muito cuidado aqui nesse ponto, o seu projeto deve estar todo numa mesma pasta. preste atenção para não colocar os arquivos em pastas diferentes. Este é um erro muito comum aos iniciantes.



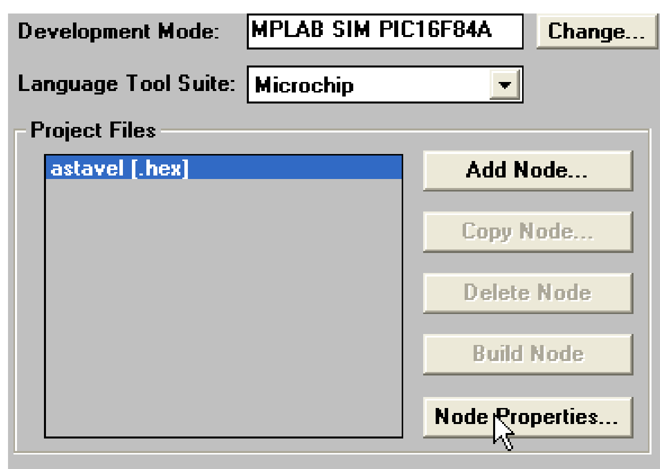
Passo 4.4

Agora apareceu essa janela do Edit Project, aonde o "arquivo alvo" o Target Filename vem com o nome que demos ao projeto, seguido da extensão .hex, que é o nosso objetivo: gerar um código hexadecimal para ser gravado no PIC. Aqui vamos dar informações preciosas do nosso projeto, quem é o fonte, que tipo de arquivo hexadecimal ele vai gerar etc. Vá lá em baixo na janela Project Files, e clique em astavel.hex. Isso só vai selecionar o arquivo, o MPlab chama cada arquivo de node.



Passo 4.5

Com o astavel.hex selecionado clique em Node Properties..

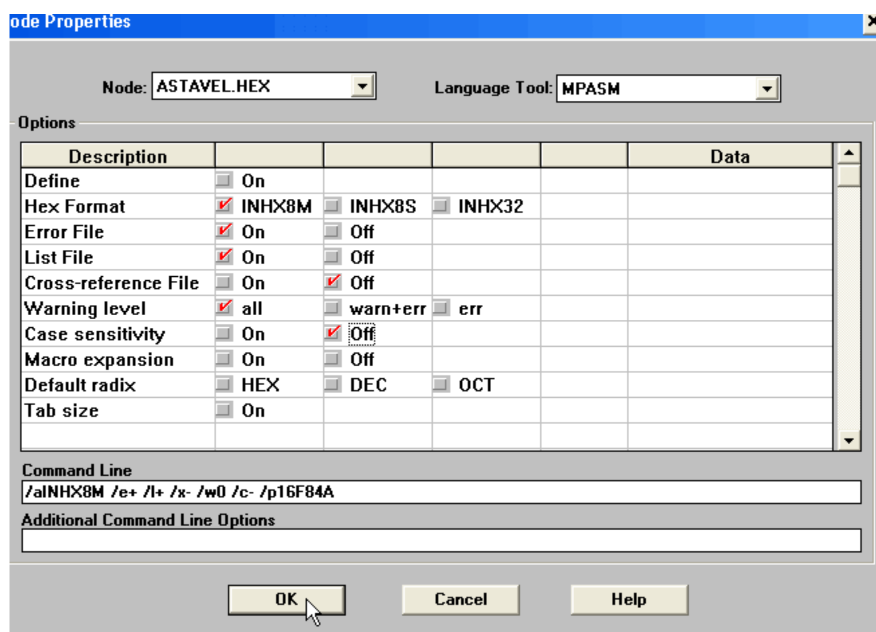


Passo 4.6

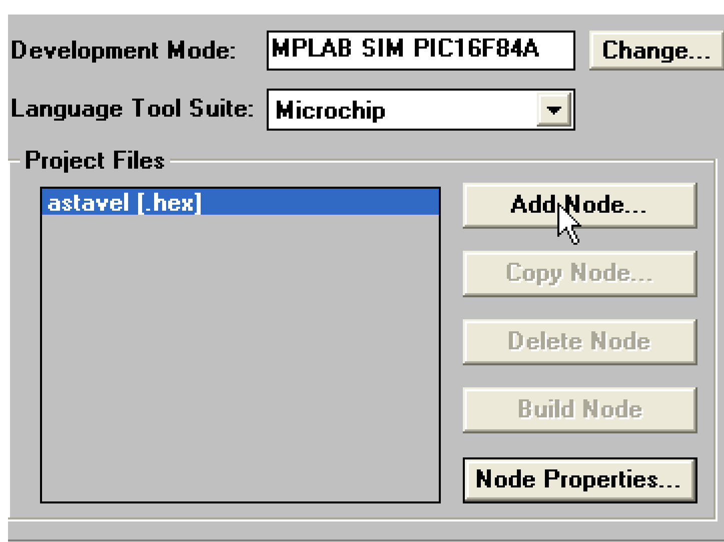
Na janela Node Properties, propriedades do node, você vai fazer 3 alterações:

1. Selecionar INHX8M (formato padrão do hexadecimal para maioria dos gravadores de pic)
2. Selecionar Warning level ALL (isso faz o compilador mostrar todos os avisos que poderiam prejudicar o seu software)
3. Selecionar Case Sensitivity OFF (isso faz com que o compilador não faça distinção entre maiúsculas e minúsculas)

Clique em OK

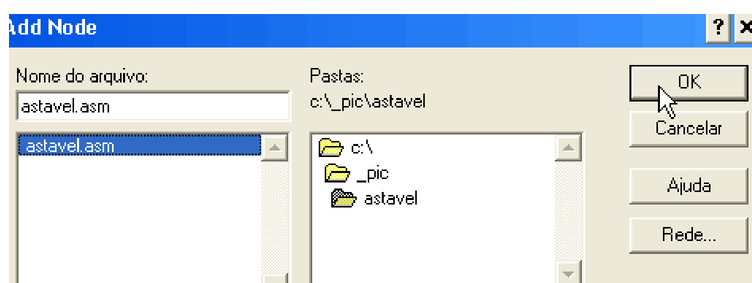


Passo 4.7
Clique em Add Node..

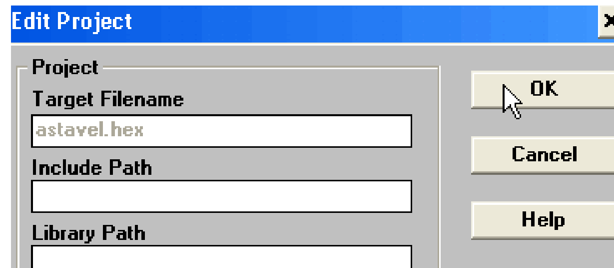


Passo 4.8

Procure na pasta astavel, selecione astavel.asm e depois clique em OK. Isso que fizemos foi adicionar o fonte ao nosso projeto. Note que isso não é uma operação automática, se você não dizer ao projeto qual o fonte utilizar, mesmo estando na mesma pasta ele não vai funcionar. Isso permite que você tenha varias versões de um código fonte, e pode adicioná-lo ao projeto em qualquer momento, editando o projeto.



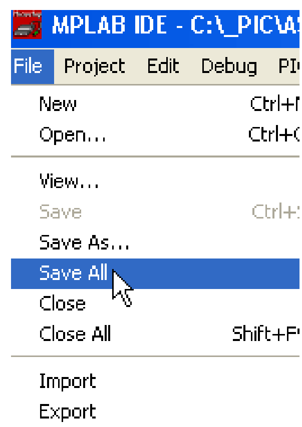
Passo 4.9
Clique em OK para confirmar as propriedades do projeto.



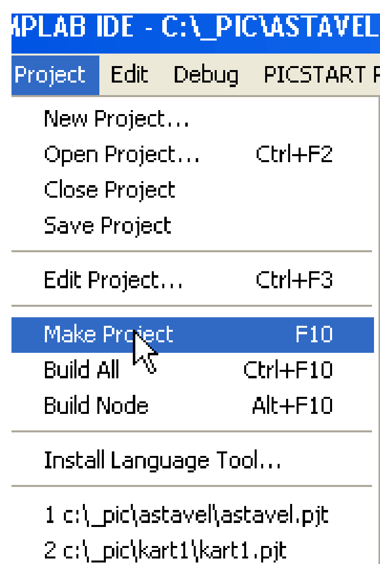
Passo 4.10

Está quase pronto! Só falta compilar: Mas antes salve tudo o que fez de vez em quando o MPlab Trava na hora de compilar... Não estranhe se isto acontecer... E se você não salvou... Tem que começar tudo de novo.

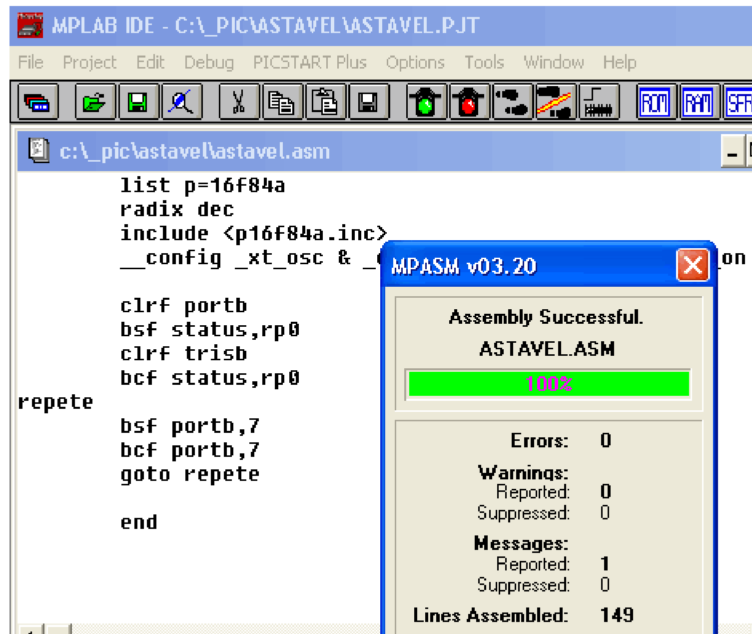
Clique em File Save All



Passo 4.11
Agora sim, vamos compilar nosso projeto! Clique em Project > Make Project



Vai aparecer momentaneamente a tela do compilador, e fecha logo após compilar.



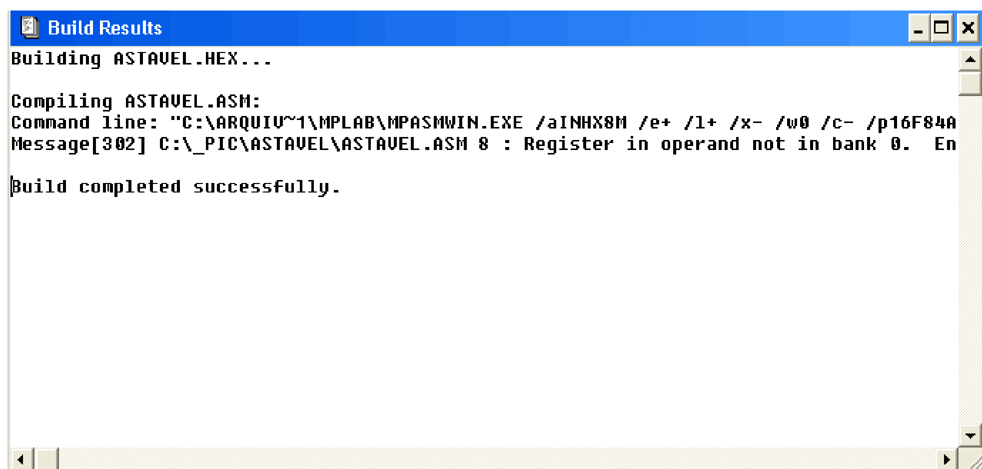
Passo 4.12

Pronto! Se você digitou tudo corretamente deve receber a seguinte Janela de resultados, o Buid Results com a seguinte frase em baixo: Buid completed successfully, ou seja foi compilado com sucesso. se você digitou alguma coisa errada ele vai escrever qual foi o erro, em inglês é claro, e vai escrever:

MPLAB is unable to find output file "ASTAVEL.HEX". This may be due to a compile, assemble, or link process failure.

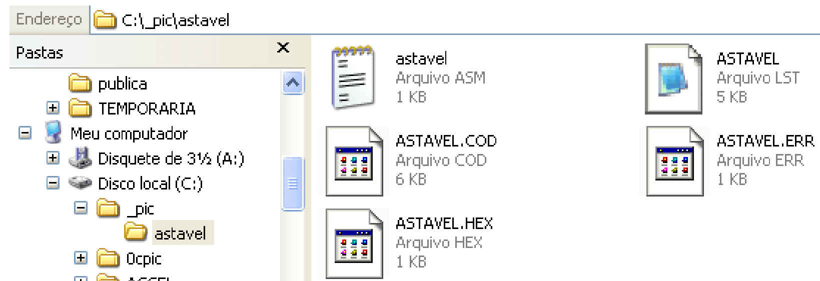
Build failed.

Se isto ocorrer temos que descobrir o erro conforme as dicas das mensagens, na próxima aula vou ensinar uma técnica para procurar os erros.

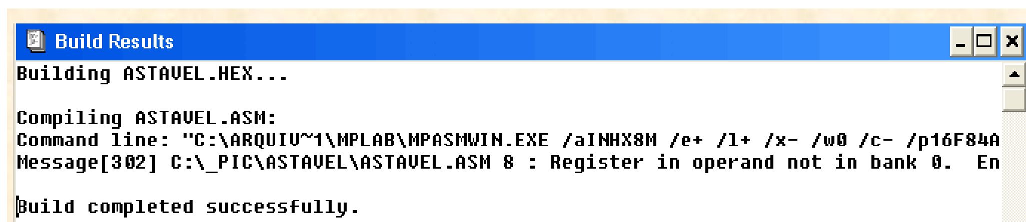


Passo 4.13

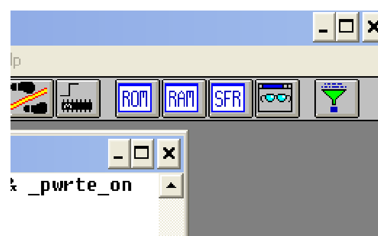
Se conseguiu chegar até aqui com a janela acima, você terminou o projeto. O Mplab gerou pra você 5 arquivos, sendo que o principal é que tem a extensão .hex, que vai ser usado para gravar o PIC. Verifique os arquivos: abra-os com o bloco de notas e dê uma olhada. (não altere nada nos arquivos, só olhe)

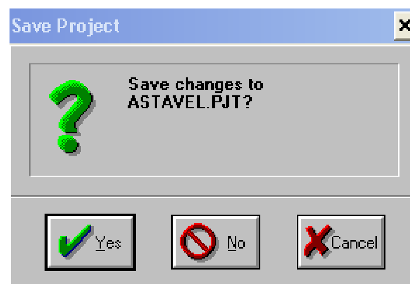


Passo 4.14 "Feche o MPlab:"
Primeiro Feche a janela Build Results (Clique no x)



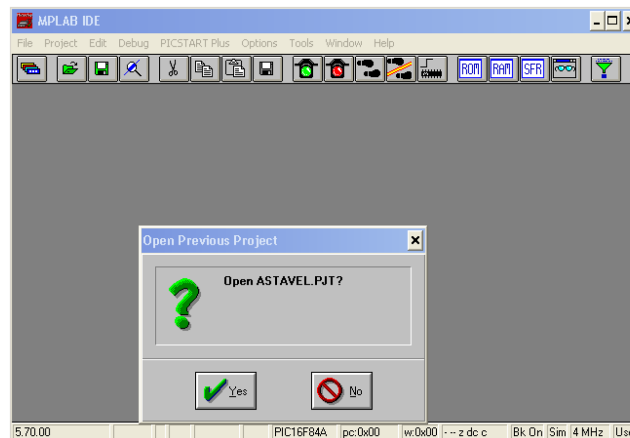
Agora Feche o MPlab. Não feche a janela do código fonte, Quando fechamos o Ambiente Integrado de desenvolvimento, o MPlab, ele se encarrega de fechar todas as janelas e guarda onde elas estavam abertas, assim a próxima vês que você abrir o Programa ele reabre todas pra você. Clique apenas no X e responda Yes para salvar as configurações do projeto (astavel.pjt).



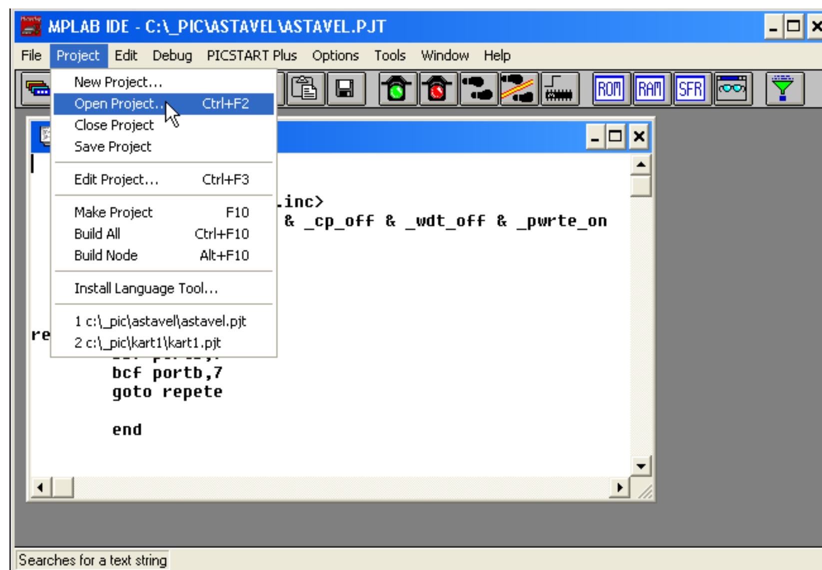


Erros de Sintaxe no Código Fonte

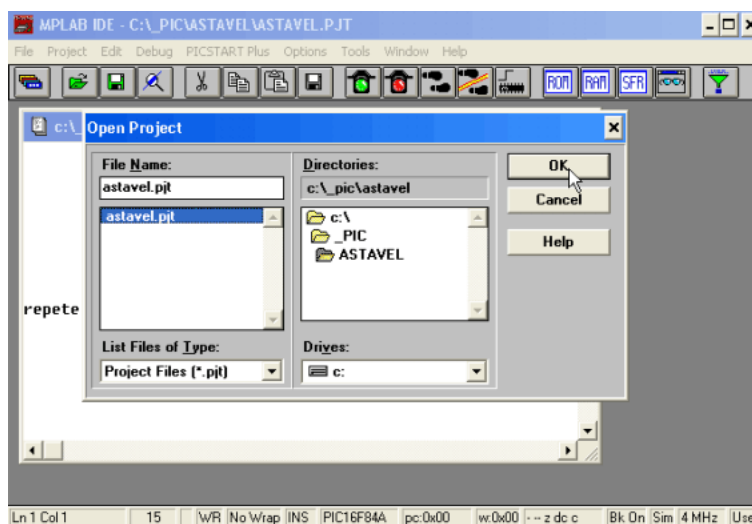
Para iniciar esta aula você deve estar com o Ambiente Integrado de Desenvolvimento MPlab IDE aberto no seu micro, com o projeto da aula anterior.Clique no icone do MPlab na área de trabalho, que você criou, ou no Menu Iniciar > Programas . O MPlab vai abrir uma caixa de diálogo perguntando se você quer abrir o último projeto trabalhado, no nosso caso o último foi o astavel.pjt, então responda YES.



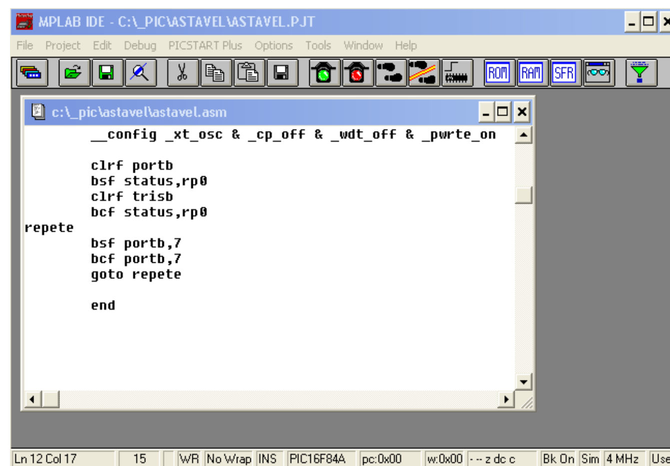
Caso não seja esse, o astavel.pjt, responda NO e vá em Project > open



Selecione o astavel.pjt no diretório que foi criado, e clique em OK



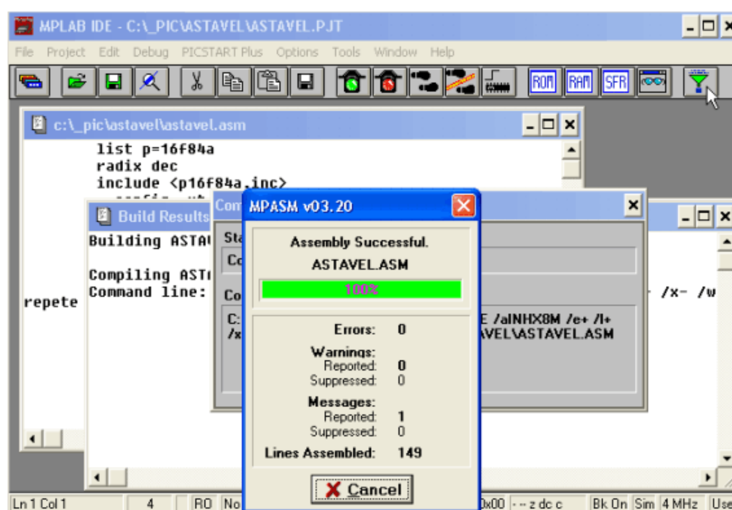
Pronto! MPlab aberto com o Projeto alvo da nossa aula:

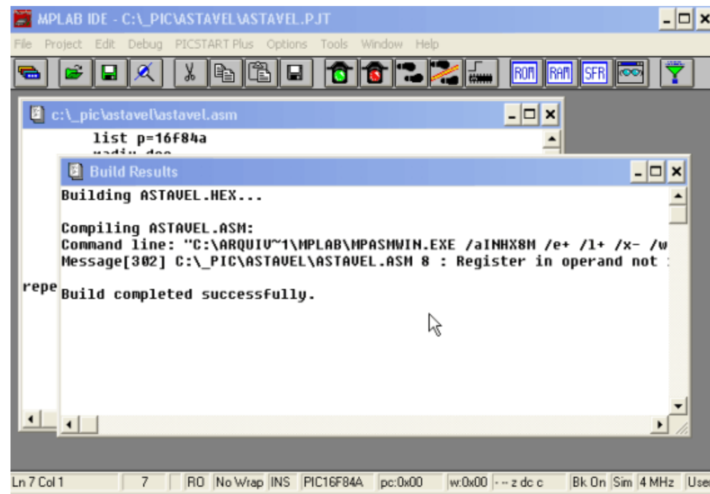


Compilando o fonte sem erros de sintaxe:

Compilar o programa significa transformar a linguagem texto do fonte em códigos hexadecimais, ou seja códigos de máquina para ser gravado no PIC. Para isso é necessário que não haja erros no programa fonte, erros de sintaxe, (escrita) ou outros erros. O erro de sintaxe, é aquele que é provocado por escrever-mos instruções ou argumentos de instruções de maneira incorreta, ou seja, não é válida, nos arquivos do MPlab aquela sequência de caracter não existe. Na aula anterior fizemos o nosso primeiro projeto, e escrevemos um código fonte para ele. a última etapa foi compilar o projeto se tudo estava correto, apareceu no MPlab uma janela de avisos, o Buid Result como abaixo.

Para compilar basta teclar F10. ou no Menu Project depois Make Project ou ainda no icone que parece um funil. Se não houver erros aparecerá uma indicação na janela do compilador dizendo que a compilação foi feita com sucesso (tudo OK) " build completed successfully" veja figura abaixo.





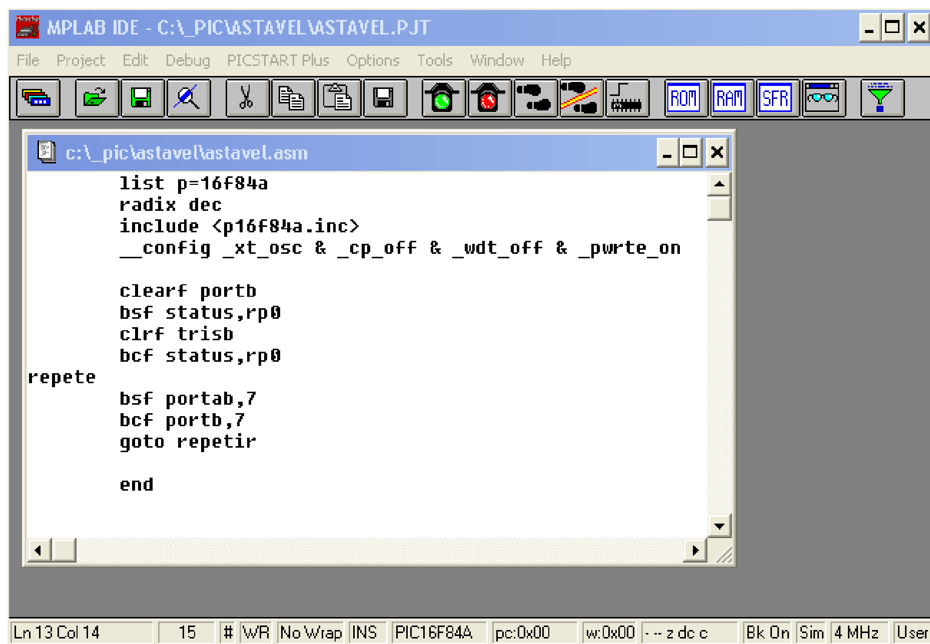
Na Janela acima vemos o seguinte: A primeira, a segunda e a terceira linha aparece nessa janela quando inicia o processo de compilação, as próximas linhas vão aparecendo conforme o progresso de compilação, depende da velocidade do processador do seu micro, normalmente é bem rápido. As linhas que vem escritas Message[xxx] são mensagens de alerta que o compilador envia para você, não são erros, no caso acima,

{Message[302] C:_PIC\ASTAVEL\ASTAVEL.ASM 8 : Register in operand not in bank 0. Ensure that bank bits are correct.} está avisando que no nosso fonte, exatamente na linha 8 o banco de memória não é o Zero. assegure-se que isto esteja correto.

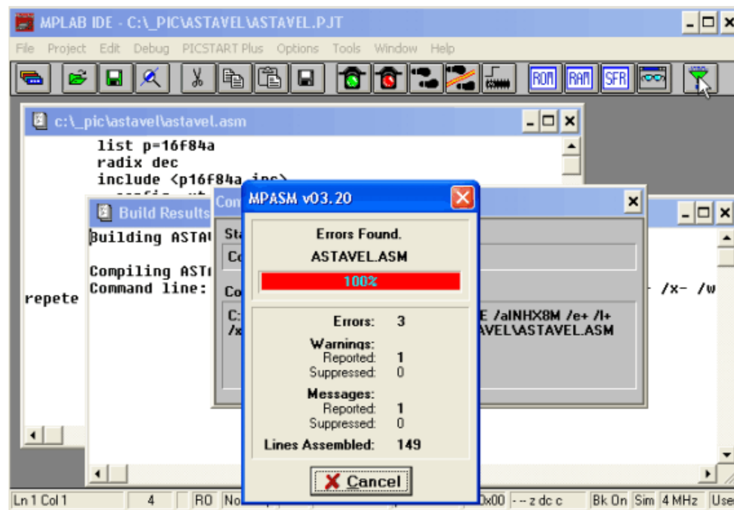
E finalmente a última linha dizendo que a compilação foi completada com sucesso.Um detalhe muito importante é que não há erro de sintaxe, mas ninguém garante aí, que não há erros de lógica do seu programa.

Vamos inserir erros no nosso fonte:

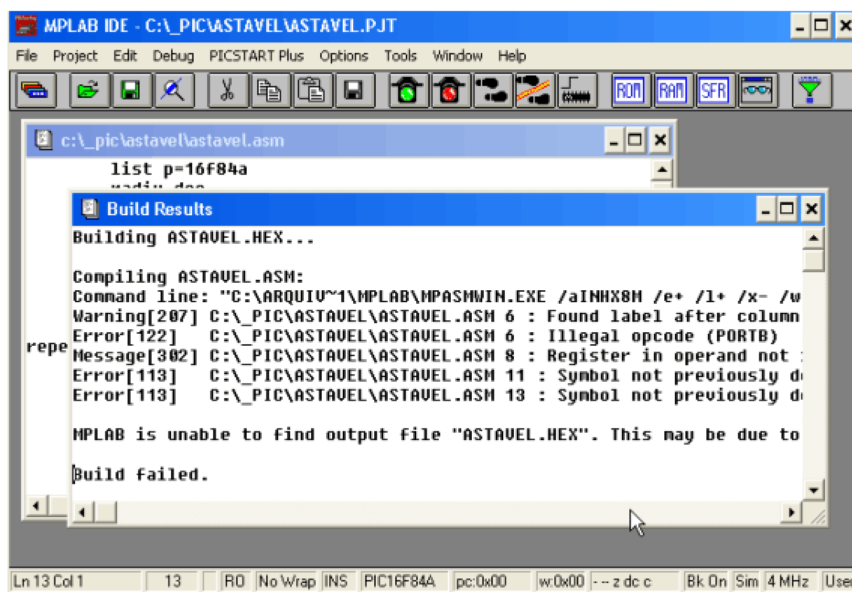
Vá no editor do código fonte, e na linha 6 mude clrf para clearf; na linha 11 mude portb para portab; na linha 13 mude repete para repetir



Vamos então Compilar nosso projeto, agora com os erros: clique no funil verde



Veja a janela de resultados:



Pronto! vamos interpretar os resultados. Nesse momento é muito importante você saber interpretar os resultados, o inglês nessa hora é essencial:

- Warning (não são erros que impedem a compilação, são apenas alertas, o compilador te avisa nesse caso de possíveis erros é bom sempre verificar) no nosso caso:

Warning[207] C:_PIC\ASTAVEL\ASTAVEL.ASM 6 : Found label after column 1. (CLEARF)

Nesse caso, o compilador esta te avisando que ele encontrou na linha 6, uma palavra, ou seja, uma string que não está na coluna 1, e que possivelmente você tivesse tentando colocar aí um label, rotulo. Não é o nosso caso, pois simulamos aí um erro na instrução clrf, bom... mas o compilador não sabe disso... e mesmo assim detectou um possível erro.

- Error (esse sim são erros na sintaxe, de instrução ou argumentos, ou outros)

Error[122] C:_PIC\ASTAVEL\ASTAVEL.ASM 6 : Illegal opcode (PORTB)

Esse Erro está informando que na linha 6 tem um código de operação ilegal, ou seja, um código de instrução que não existe no PIC. Ops... mas portb não existe? Existe sim, mas não como instrução, e sim como argumento de uma instrução, e como clearf não é

instrução, o compilador achou que era um label fora da coluna 1, está aguardando para próxima string uma instrução, ou opcode.

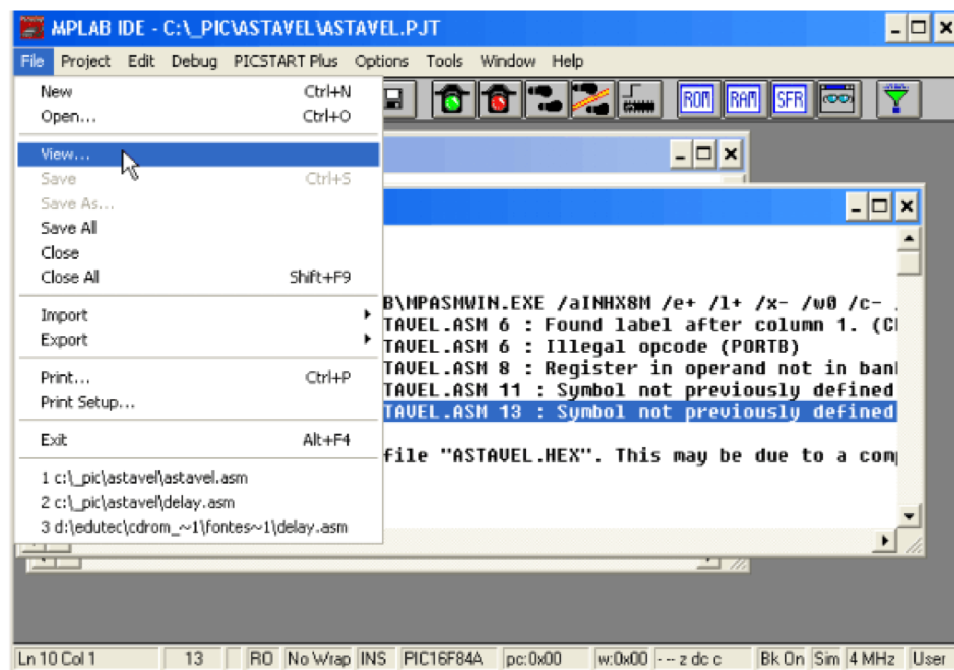
Message[302] C:_PIC\ASTAVEL\ASTAVEL.ASM 8 : Register in operand not in bank 0. Ensure that bank bits are correct.
este tipo de mensagen nós já vimos... é só um aviso que a RAM não está no banco zero nessa linha

Error[113] C:_PIC\ASTAVEL\ASTAVEL.ASM 11 : Symbol not previously defined (PORTAB)

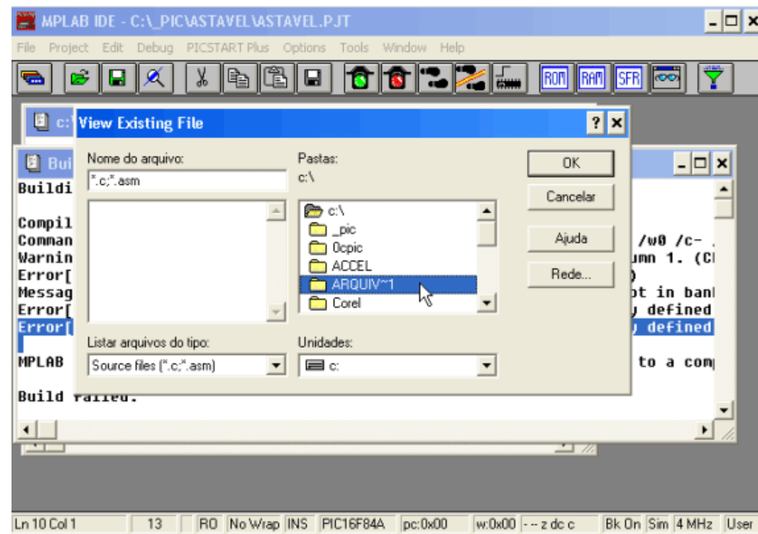
Esse erro avisa que na linha 11 existe um símbolo, ou argumento que não foi previamente definido. no caso o arquivo que mostra as definições dos símbolos, é o que colocamos no include, no caso o p16f84a.inc. vejamos este arquivo para nos familiarizarmos com os simbolos previstos pela microchip.

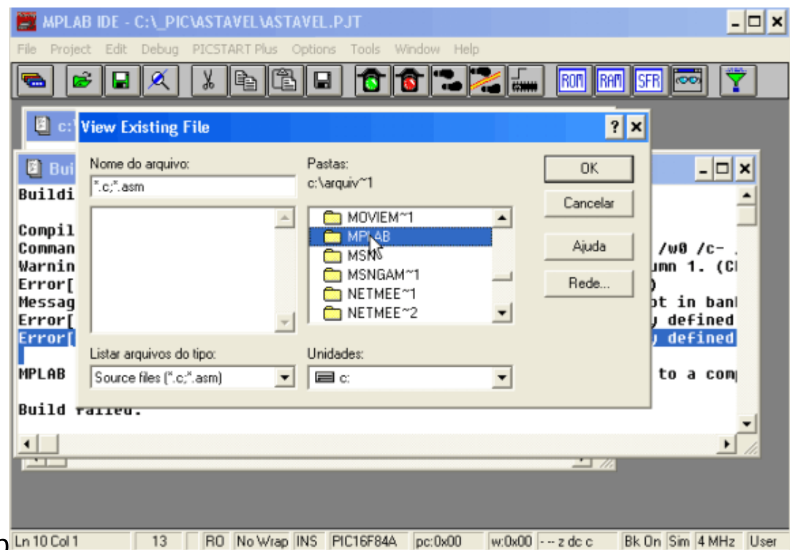
- Vendo o arquivo de equivalências de símbolos da Microchip

Vá em File > View (essa opção o arquivo só se torna só leitura, isso evita fazermos alterações indesejadas)



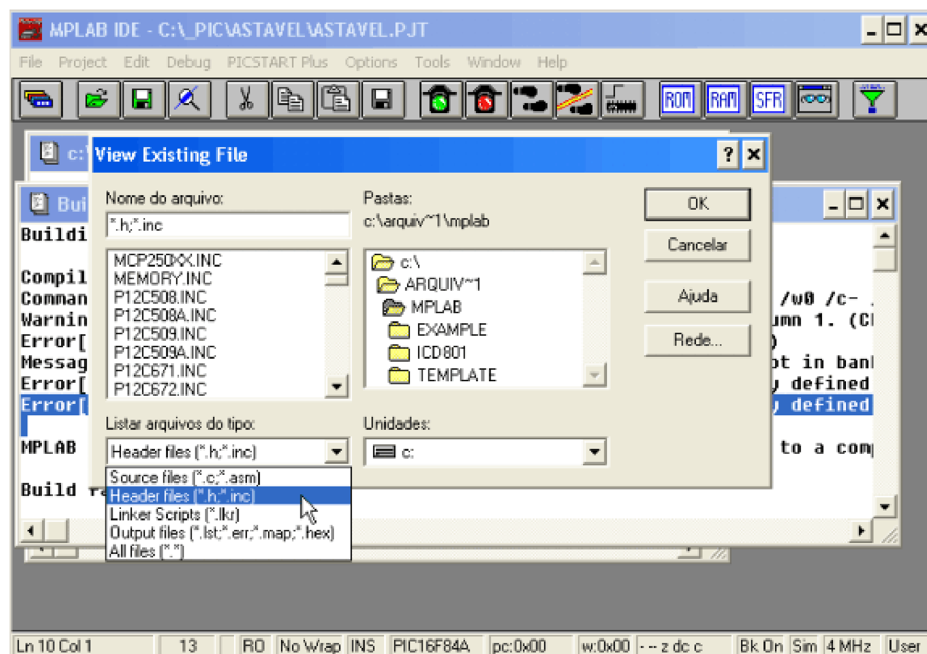
Procure o diretório onde o Mplab foi instalado. normalmente é arquivo de programas



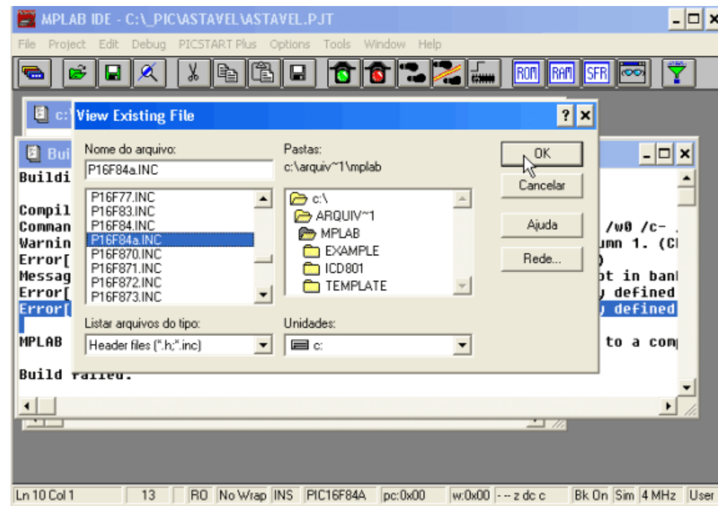


arquivos de programa > Mplab

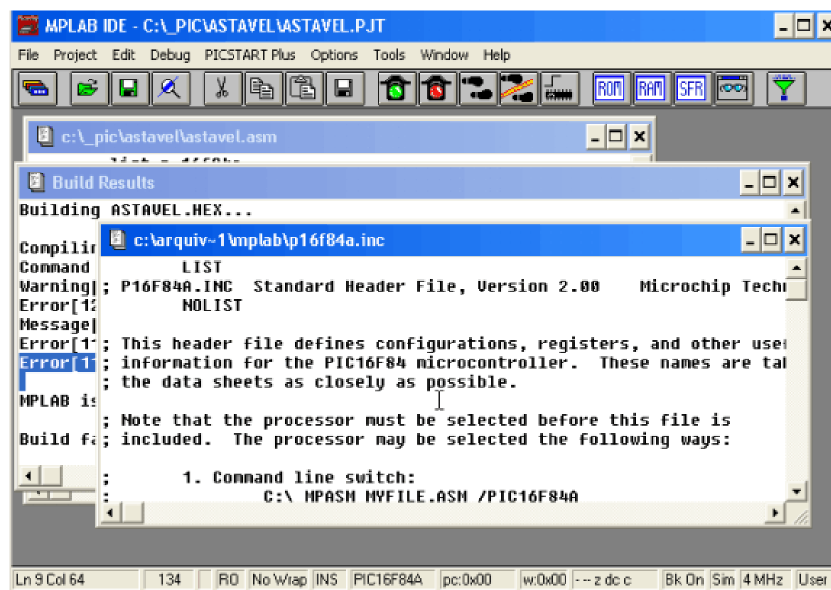
Altere a lista de tipos de arquivos, para "h" inc



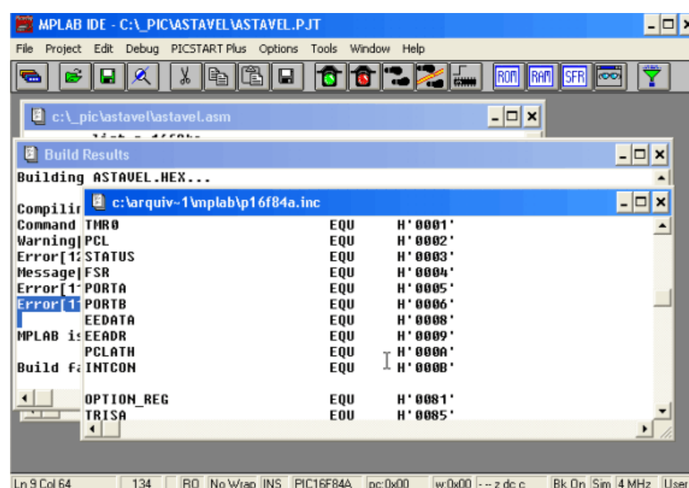
Selecione o p16f841.inc e clique em OK



O arquivo está pronto para você olhar... navegue pelo arquivo.



Veja os simbolos que Microchip usa para a programação dos PIC´s



Nesse caso, quando digitamos PORTB o compilador vai trocar por 6, pois no arquivo acima está escrito que PORTB EQUivale a 6 (EQU > equate). O Bom disso, é que você pode criar os seu símbolos personalizados, exemplo: em vêz de portb você pode colocar SAIDA2... etc.

- Voltando aos erros

Error[113] C:_PIC\ASTAVEL\ASTAVEL.ASM 13 : Symbol not previously defined (REPETIR)

Esse erro avisa que na linha 13 existe um outro simbolo que não foi previamente definido, no nosso caso o label definido na coluna 1 da linha 10 é o repete e não repetir.

Técnicas para solucionar os erros:

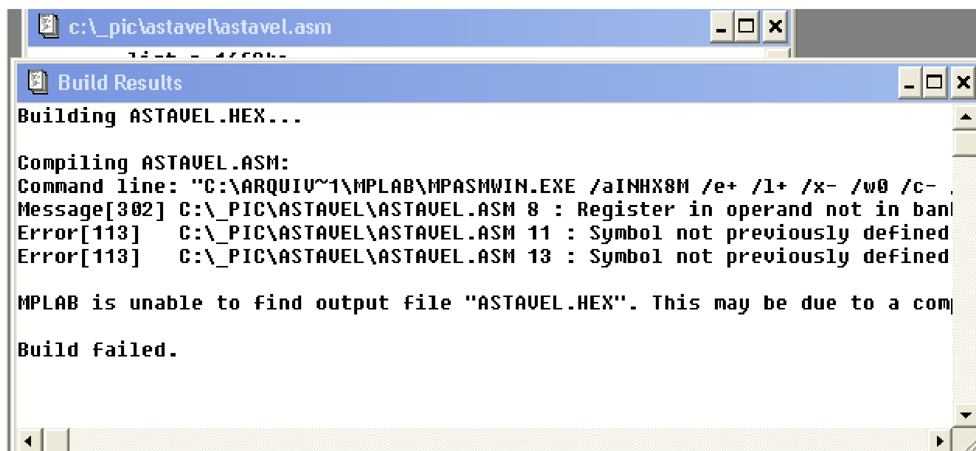
1. Começar sempre pelo primeiro warning ou erro detectado. Isso ajuda pois às vezes um problema no começo do programa pode gerar erros em muitas linhas abaixo dele.
2. A cada erro solucionado faça um a nova compilação. Aí você pega o primeiro erro de novo, e assim por diante até solucionar todos.
3. Se o erro for Símbolo não definido previamente, verifique o .inc, isso pode ajudar. É comum esquecer o include... aí quase tudo é símbolo não definido previamente.

DICA: CLIQUE DUAS VEZES NA LINHA DO ERRO DO BUILD RESULT, QUE O PRÓPRIO MPLAB VAI ABRIR O ARQUIVO FONTE, E O CURSOR VAI EXATAMENTE NA LINHA DO ERRO.

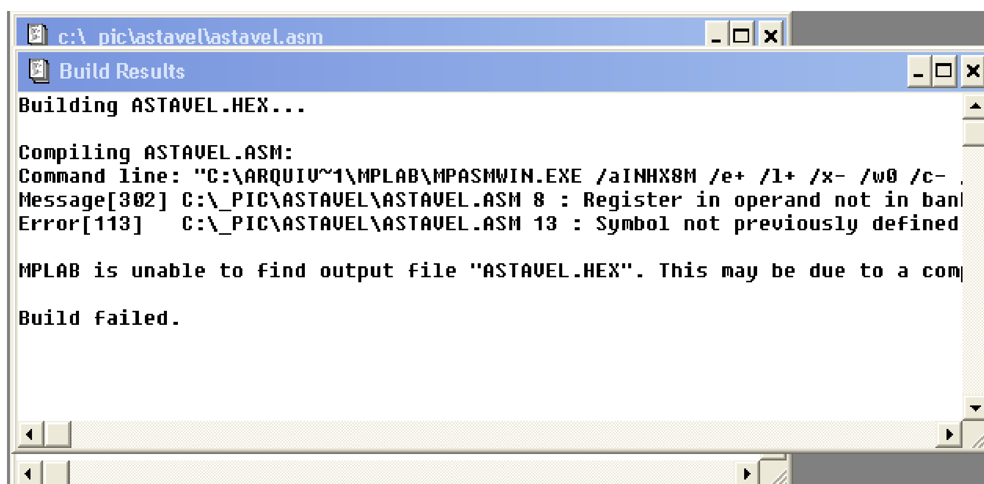
Feche o Visualizador do arquivo p16f84a.inc, Solucione os problemas, da forma descrita:

Passo 01 (clique 2 vezes no Warning[207] e mude de clearf para clrf)

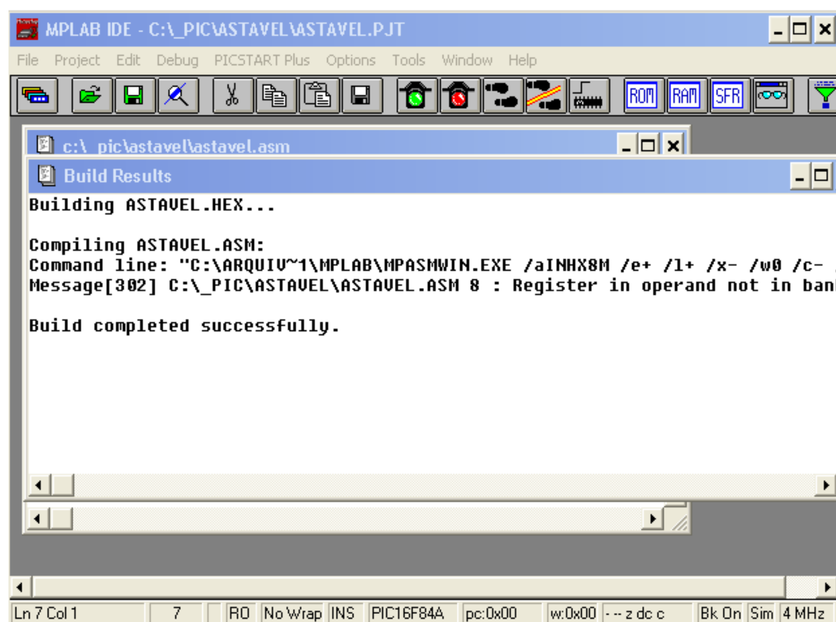
Compile apertando F10 ou clicando no funil.



Passo 02 (clique 2 vezes no primeiro Error[113] e mude de portab para portb)
Compile apertando F10 ou clicando no funil.



Passo 03 (clique 2 vezes no Error[113] e mude de repetir para repete)
Compile apertando F10 ou clicando no funil.



Pronto! Compilado com Sucesso novamente.

Feche o Buid Results, o MPlab salvando as configurações ao sair, como descrito no final da aula04.

EXERCÍCIOS

1. De acordo com que você aprendeu até agora, Descreva passo a passo o procedimento para criar um novo projeto com o nome de tarefa01.pjt.
2. Crie uma pasta tarefa01 dentro do diretório _PIC
3. Abra o MPlab respondendo NO quando ele perguntar se você quer abrir o último projeto.
4. Edite o fonte abaixo e salve como tarefa01.asm na pasta tarefa01 que você criou no exercício 2

```
list p=16f84a
radix dec
include <pic16f84a.inc>
__config _xt_osc & _cp_off & _wdt_off & _pwrte_on

clrf portb
bsf statos,rp0
clrf trisb
bcf status,rp0
```

5.

```
repete
```

6.

```
bsf portb,7
bcf portb,7
repete
```

7. end
8. Faça o projeto como descrito na aula04: Project >New ....
9. Compile o projeto. Vai ter um montão de erros... Encontre os erros e corrija.
10. vai compilando até ter sucesso.
11. Mande-me através do nosso canal de comunicação, os erros que você encontrou, dizendo a linha e qual foi o erro.

Bons estudos...

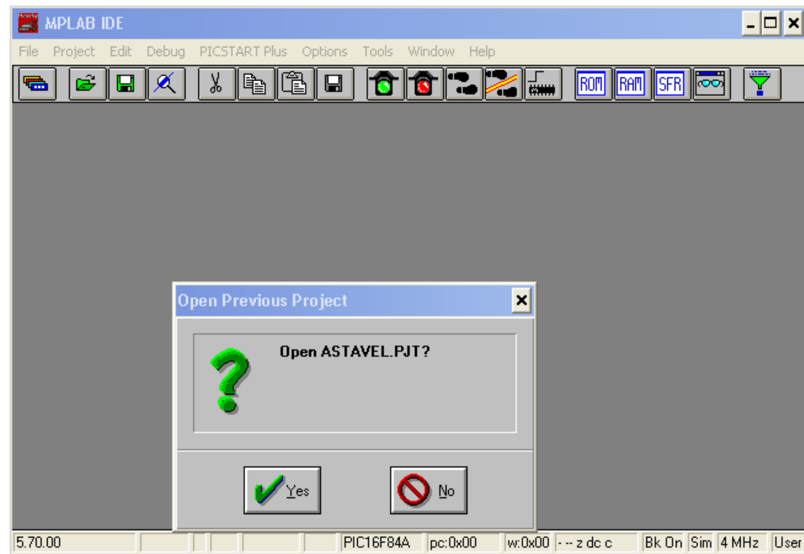
Abrindo um projeto já existente

Nesta aula vamos aprender a abrir e editar um projeto já existente. Isso ocorre quando queremos trabalhar num projeto que já está em andamento, ou quando precisamos fazer algumas mudanças em projetos já acabados.

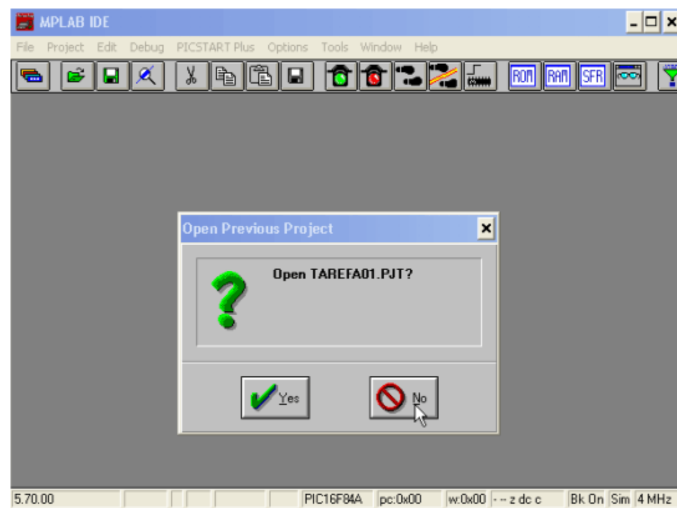
Como já comentamos nas primeiras aulas, é importate a organização de seus projetos, sendo um projeto em cada pasta, e todas as pastas num único diretório para facilitar. Nós já temos nesse momento dois projetos: o astavel e a tarefa01. Vamos abrir o astavel:

PASSO 01

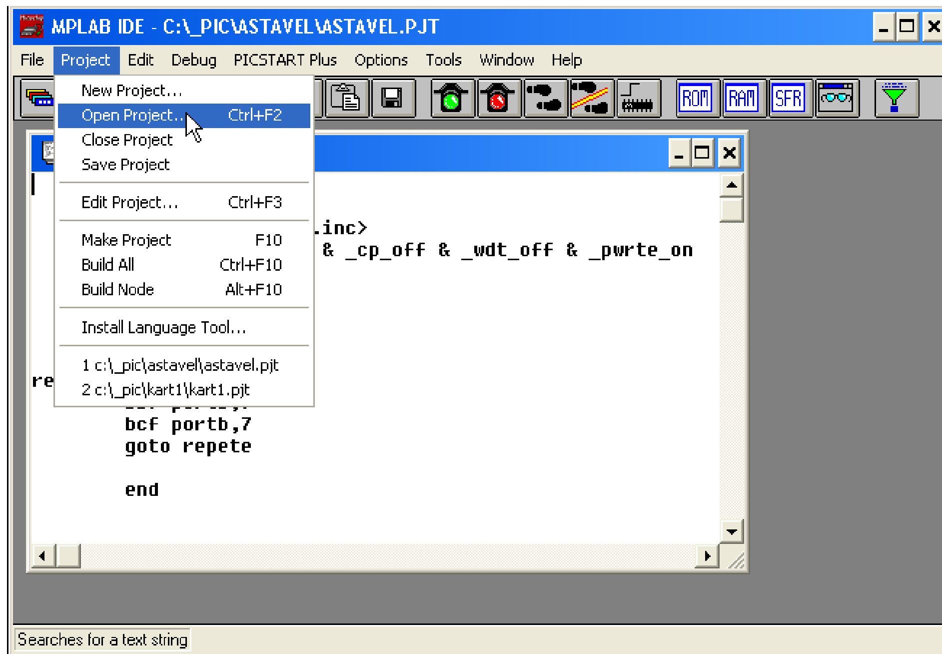
Clique no icone do MPlab na área de trabalho, que você criou, ou no Menu Iniciar > Programas . O MPlab vai abrir uma caixa de diálogo perguntando se você quer abrir o último projeto trabalhado, no meu caso o último foi o astavel.pjt, então respondo YES;



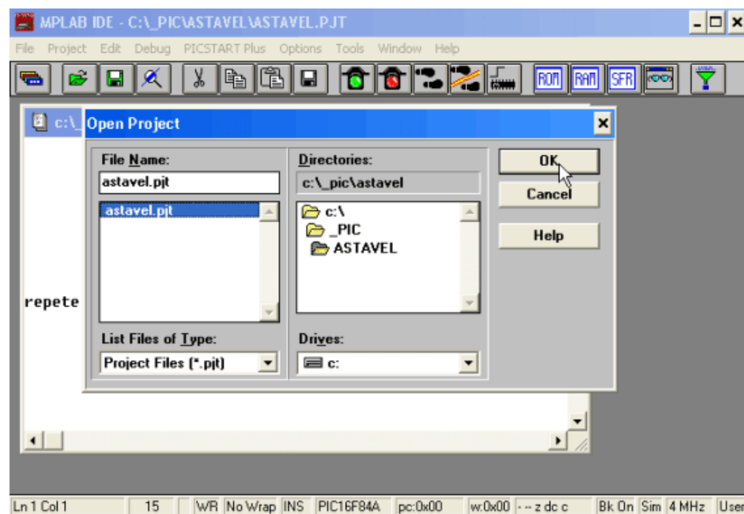
Caso não seja esse, responda NO.



Vá em Project > Open Project

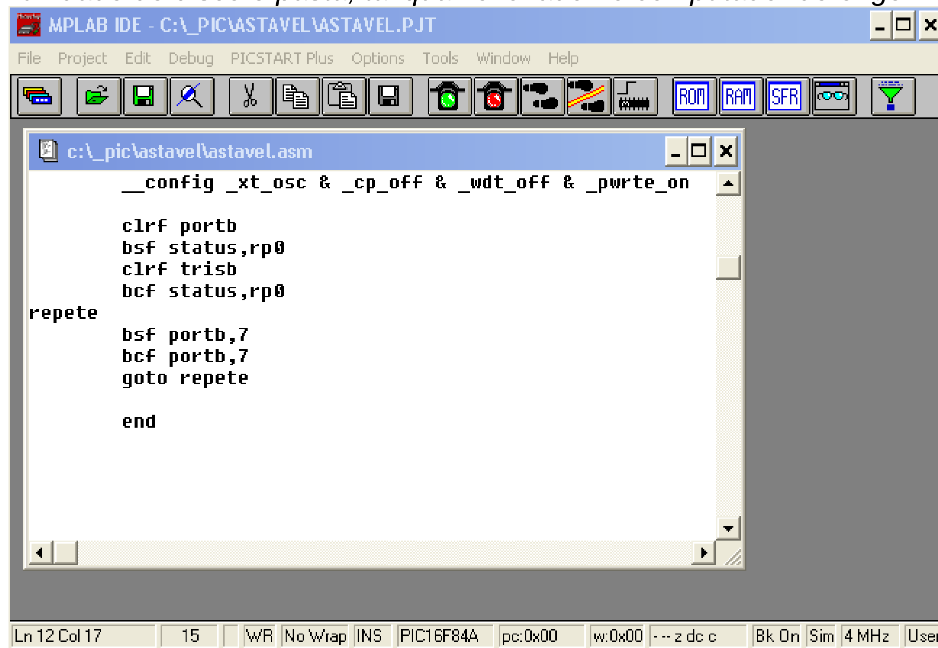


Selecione o astavel.pjt no diretório que foi criado, e clique em OK



Pronto! MPlab aberto com o Projeto astavel que já era existente aberto. para abrir qualquer outro projeto já existente siga sempre estes passos. *Observação Importante se por acaso você estiver copiando um projeto de um computador para outro, tome sempre o cuidado de copiá-lo na mesma*

unidade de disco e pasta, tal qual foi criado no computador de origem.

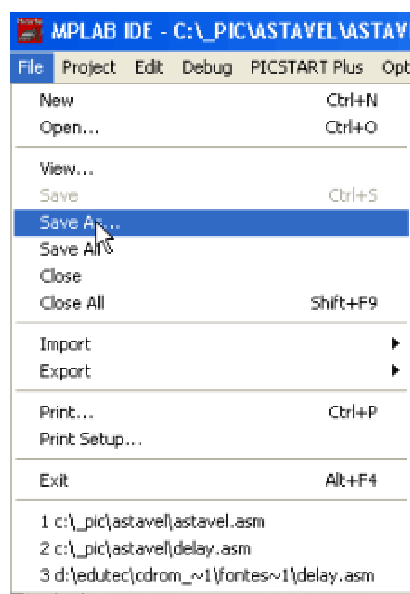


Editando um projeto já existente

Há situações que exigem a necessidade de se fazer alterações nos projetos já existentes. Por exemplo uma atualização da versão do código fonte, uma melhoria no software ou simplesmente uma correção de falha. Para você não perder a versão anterior é aconselhavel salvar o fonte com outro nome, no nosso caso poderia se chamar astave_1, lembrando que o MPlab traz uma limitação do DOS, onde o nome dos arquivos não devem ultrapassar 8 caracteres. Aí nós alteramos o fonte astave_1 com as atualizações. Mas o nosso projeto está direcionado para compilar o astavel.asm e não o astave_1, então temos que edita-lo:

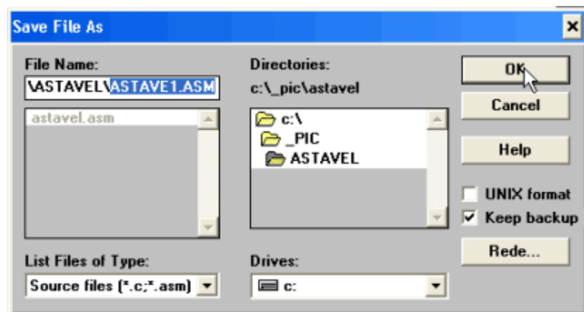
PASSO 01

Clique em File > Save As..



Passo 02

Troque o nome do arquivo de astavel.asm para astave_1.asm e clique em OK.



PASSO 03

Altere o fonte. No nosso caso vamos colocar 2 nop depois do bsf portb,7. O NOP é uma instrução do PIC que faz com que ele fique um ciclo de máquina sem fazer nada. (No OPeration). Isto no fonte vai fazer com que nosso astável gere realmente uma onda quadrada 3us em estado alto e 3us em estado baixo.

c:_pic\astavel\astave1.asm

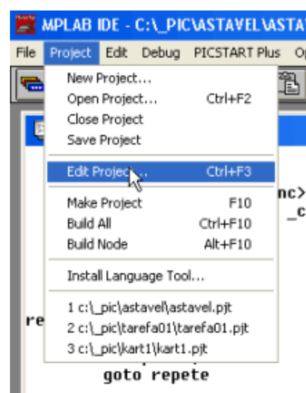
```
        list p=16f84a
        radix dec
        include <p16f84a.inc>
        __config _xt_osc & _cp_off & _wdt_off & _pwrte_on

        clrf portb
        bsf status,rp0
        clrf trisb
        bcf status,rp0
repete
        bsf portb,7
        nop
        nop
        bcf portb,7
        goto repete

        end
```

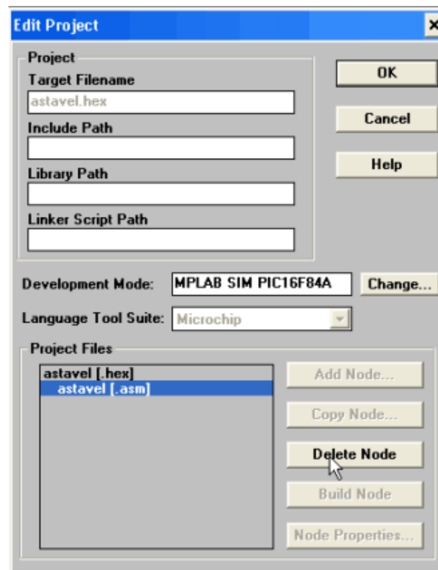
PASSO 04

Agora vamos editar o projeto para que o nosso novo fonte seje compilado. Clique em Project > Edit Project

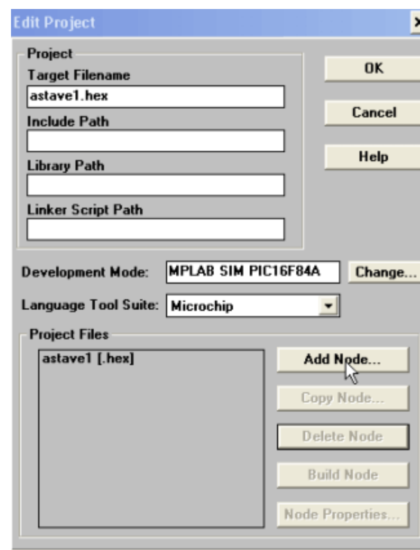


PASSO 05

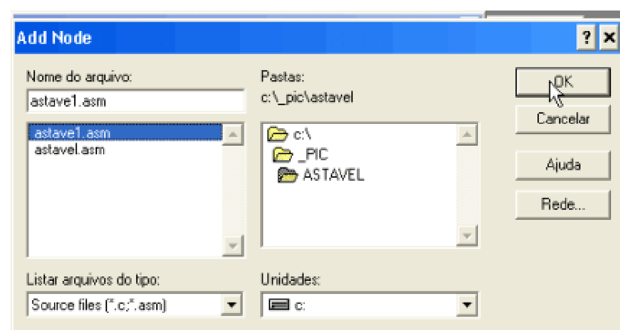
Selecione o astavel[.asm] e clique em Delete Node. Isto tira o fonte antigo do projeto



PASSO 06
Clique em Add Node,

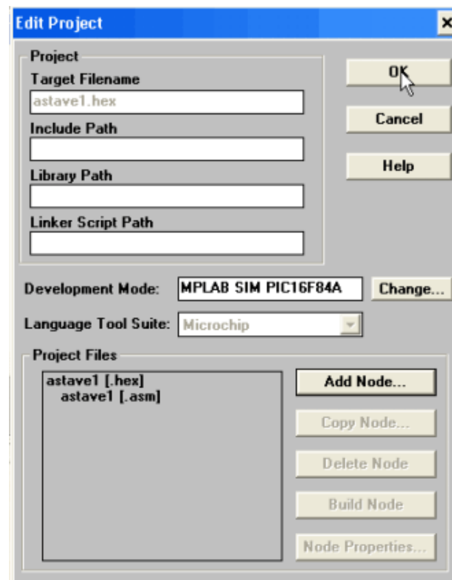


selecione o astave_1.asm e depois clique em OK.



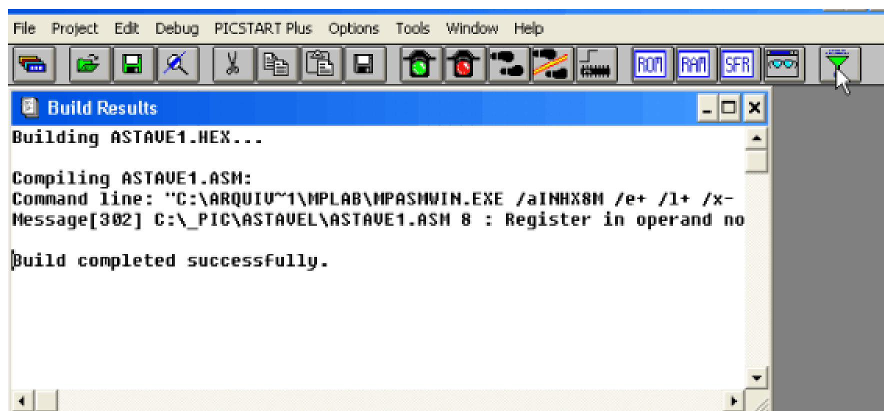
PASSO 07

Clique em OK para terminar a edição do projeto. Note que o MPlab mudou automaticamente o arquivo alvo, o hexadecimal que será gravado no PIC, ficou astave_1.hex



PASSO 08

Agora é só compilar. Clique no ícone do funil.



PASSO 09

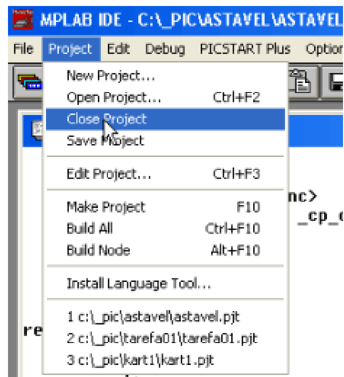
Feche a janela do Build Results e pronto.

Fechando um projeto

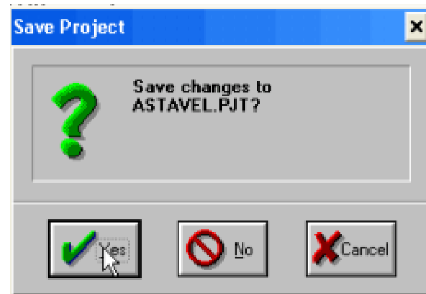
É muito importante este momento, o de fechar o projeto. Às vezes a gente fecha as janelas do Buid Results e do código fonte, e pensa que fechou o projeto... Mas ele fica alí aberto, qualquer alteração que fizermos dai para frente vai alterar tudo. É muito importante adquirir o hábito de fechar o projeto quando terminar seu uso. São duas formas para fechar: a primeira fechar só o projeto deixando o MPlab ativado para se abrir um novo projeto. A segunda forma é fechar o projeto e o MPlab ao mesmo tempo, isso significa que para abrir um outro projeto você vai ter que abrir o MPlab novamente. Analise sempre qual a melhor forma de fechar seu projeto.

FECHANDO SÓ O PROJETO ATUAL:

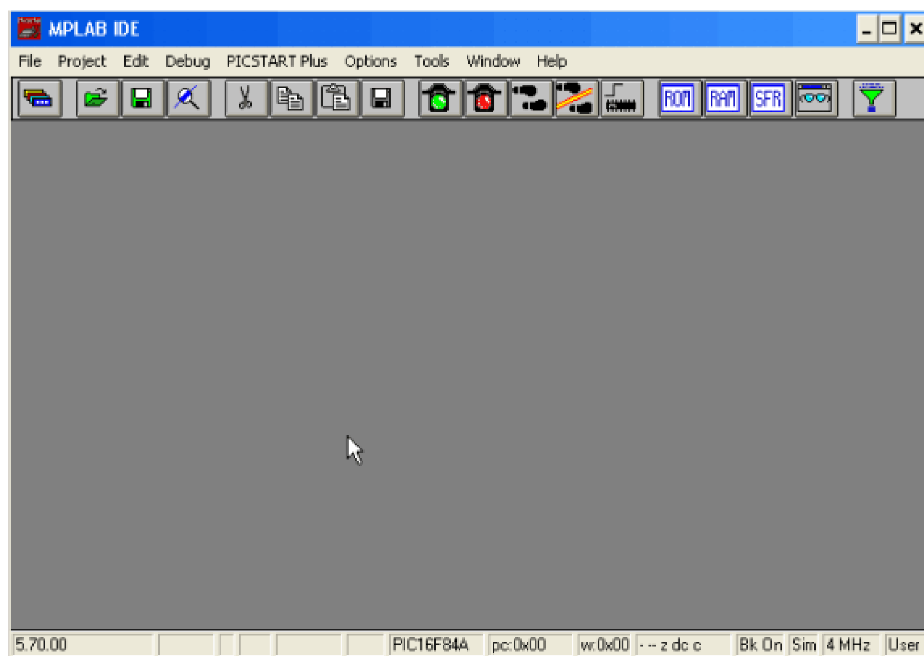
Clique Project > Close Project



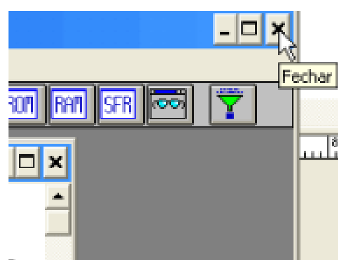
Clique em Yes para salvar as mudanças que fêz.



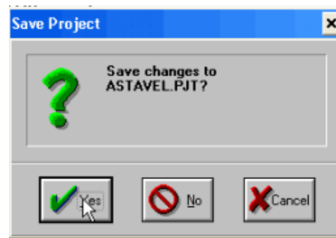
O Mplab Continuara aberto para seus outros trabalhos.



FECHANDO O PROJETO E O MPLAB AO MESMO TEMPO
Clique no X para fechar o MPlab. Não feche a janela do fonte.



Automaticamente aparecerá a Janela para você salvar seu projeto.
Clique em Yes para salvar as mudanças que fêz e pronto. oO MPlab se fecha sozinho.



EXERCÍCIOS

1. Faça outras modificações no projeto tarefa01. É muito importante o treino desses pequenos detalhes, é muito importante estar dominando o básico do MPlab para depois nos dedicarmos sómente ao assembler do PIC sem se preocupar com as operações basicas do Ambiente integrado de desenvolvimento.

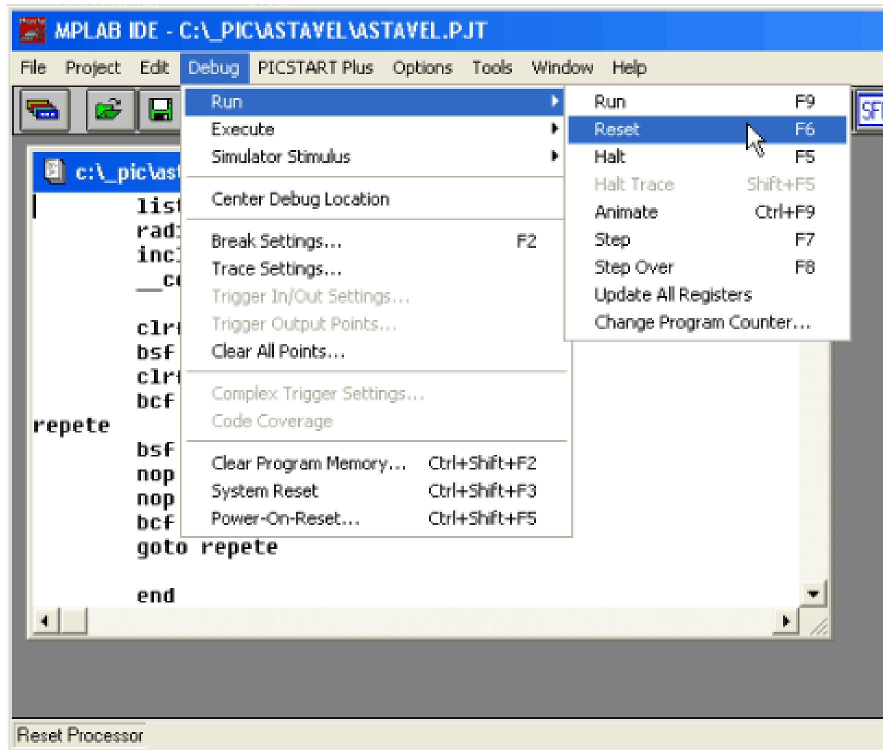
Na próxima aula iremos aprender a fazer simulações no Mplab. Treine bastante a criação de projetos.

Usando a Simulação do MPlab

O MPLAB pode simular o funcionamento do PIC. A Simulação é um recurso muito bom para depurarmos nosso programa. A simulação não ocorre em tempo real, isto é, se você fizer um programa de um temporizador por exemplo, que depois de acionado uma chave demore 15 segundos para acender um led. Isto em simulação pode levar muitos minutos, dependendo da velocidade do seu computador. Mas isso não inviabiliza a ferramenta, pois temos recursos de desviar de algumas rotinas conhecidas, ganhando esse tempo de simulação, isso nós vamos estudar numa próxima aula. Nesta aula vamos ver o básico da simulação. As ferramentas se encontram no menu Debug, Você pode ativar o comando direto no menu com o mouse, ou usar teclas de atalho. Eu prefiro usar as teclas de atalho.

Abra o astavel.pjt, lembre-se que nós fizemos uma mudança no código fonte, mantenha desse mesmo jeito.

Clique em Debug > Run > Reset, ou tecle F6



Principais teclas para a simulação

A seguir uma breve explicação dos principais comandos de simulação. Para mais detalhes consulte o manual do Mplab item 1.7 pagina 129.

- F6 Equivale ao reset da CPU. Posiciona o contador de RESET programa no endereço 0000, e coloca uma barra preta sobre a linha correspondente. Esta barra indica "a próxima" instrução a ser simulada.
- F7 A cada toque em F7 o MPLAB executa uma instrução do STEP programa. E´ como se o processador rodasse uma instrução de cada vez. Se for mantido pressionada, executará as instruções no intervalo de repetição automática da tecla.
- CTRL + F9 Roda o programa passo a passo dinamicamente, ANIMATE tornando possível acompanhar visualmente a seqüência do programa.
- F9 Realiza a simulação rápida, sem atualizar a tela. RUN Ideal para simular situações que tomariam demasiado tempo na animação.
- F5 Interrompe a simulação dinâmica iniciada pelo Ctrl+F9 STOP ou pelo F9

Você deve estar vendo a seguinte tela: a tarja preta sobre a instrução clrf, é a forma do MPlab indicar a posição da instrução a ser executada quando ativamos a simulação. Nesse caso, o Reset manda para a posição 0000 do pic, que no nosso fonte é a instrução clrf portb.

c:_pic\astavel\astave1.asm

```
        list p=16f84a
        radix dec
        include <p16f84a.inc>
        __config _xt_osc & _cp_off & _wdt_off & _pwrte_on

        clrf portb
        bsf status,rp0
        clrf trisb
        bcf status,rp0
repete
        bsf portb,7
        nop
        nop
        bcf portb,7
        goto repete

        end
```

Você pode ver o endereço onde cada instrução foi gravada no PIC, abrindo o arquivo que foi gerado durante a compilação cuja extensão é .lst. Para fazer isso de uma forma rápida e fácil, é só clicar em Window > Absolute Listing

Neste arquivo temos todas as informações do nosso programa: endereço onde foi gravado a instrução, código hexadecimal da instrução, linha que a instrução está no programa fonte, menssagens do compilador, label´s e simbolos usados, tamanho da memória que foi ocupado etc. no destaque o endererço 000. Navegue pelo arquivo depois fechhe-o.

```
c:\_pic\astavel\astave1.lst

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                      00001           LIST P=16F84A
                      00002           RADIX DEC
                      00003           INCLUDE <P16F84A.INC>
                      00001           LIST
                      00002 ; P16F84A.INC  Standard Header File, Version 2.
                      00134           LIST
2007   3FF1           00004           __CONFIG _XT_OSC & _CP_OFF & _WDT_OFF &
                      00005
0000   0186           00006           CLRF PORTB
0001   1683           00007           BSF STATUS,RP0
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are
0002   0186           00008           CLRF TRISB
0003   1283           00009           BCF STATUS,RP0
0004                  00010 REPETE
0004   1786           00011           BSF PORTB,7
0005   0000           00012           NOP
0006   0000           00013           NOP
0007   1386           00014           BCF PORTB,7
0008   2804           00015           GOTO REPETE
                      00016
                      00017           END
```

Simulando passo a passo (F7)

Certifique-se que a janela do arquivo fonte tenha o foco do windows, foco é a janela ativa, o padrão windows é o título da janela estar tarjado de azul, para confirmar é só clicar com o mouse dentro da janela do fonte. Por enquanto vamos apenas ver a sequência que as instruções sao executadas, depois veremos os registros internos do pic sendo modificados.

```
File  Project  Edit  Debug  PICSTART Plus  Options  Tools  Window  Help

c:\_pic\astavel\astave1.asm
        list p=16f84a
        radix dec
        include <p16f84a.inc>
        __config _xt_osc & _cp_off & _wdt_off & _pwrte_on

        clrf portb
        bsf status,rp0
        clrf trisb
        bcf status,rp0
repete
        bsf portb,7
        nop
        nop
        bcf portb,7
        goto repete

        end
```

Agora clique varias vezes em F7 e acompanhe o que acontece com a tarja preta, cada clique ela executa a instrução que estava tarjada e vai pra próxima

c:_pic\astavel\astave1.asm

```
        list p=16f84a
        radix dec
        include <p16f84a.inc>
        __config _xt_osc & _cp_off & _wdt_off & _pwrte_on

        clrf portb
        bsf status,rp0
        clrf trisb
        bcf status,rp0
repete
        bsf portb,7
        nop
        nop
        bcf portb,7
        goto repete

        end
```

c:_pic\astavel\astave1.asm

```
        list p=16f84a
        radix dec
        include <p16f84a.inc>
        __config _xt_osc & _cp_off & _wdt_off & _pwrte_on

        clrf portb
        bsf status,rp0
        clrf trisb
        bcf status,rp0
repete
        bsf portb,7
        nop
        nop
        bcf portb,7
        goto repete

        end
```

Veja aqui que interessante: a instrução manda ir para o label repete, clicando em F7 a tarja preta voltará para a posição logo após o label repete.

c:_pic\astavel\astave1.asm

```
        bsf status,rp0
        clrf trisb
        bcf status,rp0
repete
        bsf portb,7
        nop
        nop
        bcf portb,7
        goto repete

        end
```

Agora isso fica se repetindo eternamente. (laço eterno)

c:_pic\astavel\astave1.asm

```
        include <p16f84a.inc>
        __config _xt_osc & _cp_off & _wdt_off & _pwrte_on

        clrf portb
        bsf status,rp0
        clrf trisb
        bcf status,rp0
repete
        bsf portb,7
        nop
        nop
        bcf portb,7
        goto repete

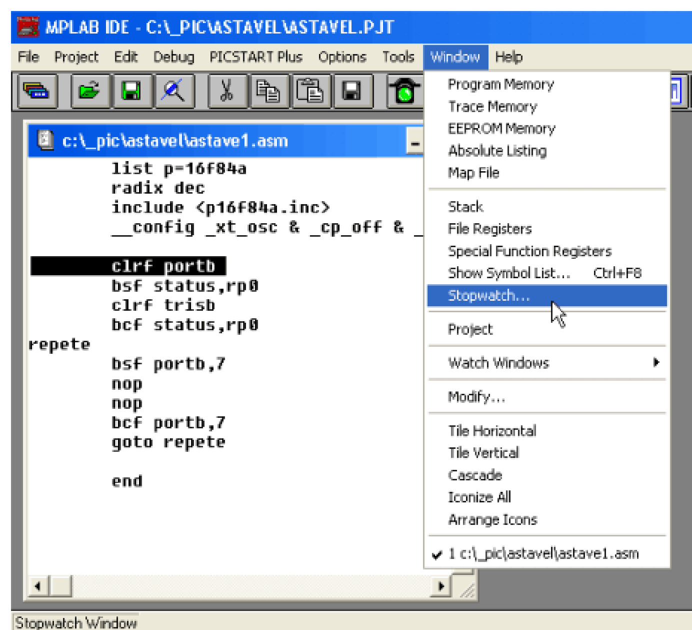
        end
```

Verificando os registros internos do PIC.

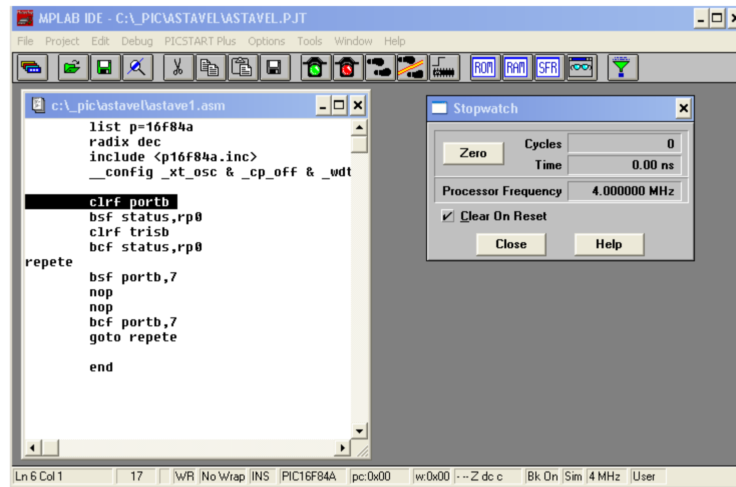
O Simulador possui algumas janelas de observações, vamos ver duas: a Stopwatch e a de observação dos registros.

- Janela Stopwatch Esta janela nos proporciona verificar o tempo decorrido de cada instrução, quantos ciclos de máquinas, com a possibilidade de zerar a qualquer momento para verificarmos o tempo exato de um determinado trecho de programa.

Clique em Window > Stopwatch

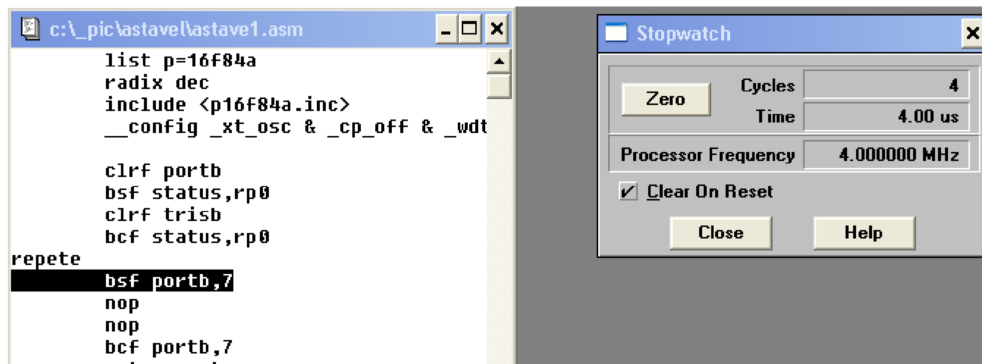


Ajuste o tamanho das janelas abertas para você poder observar todas.



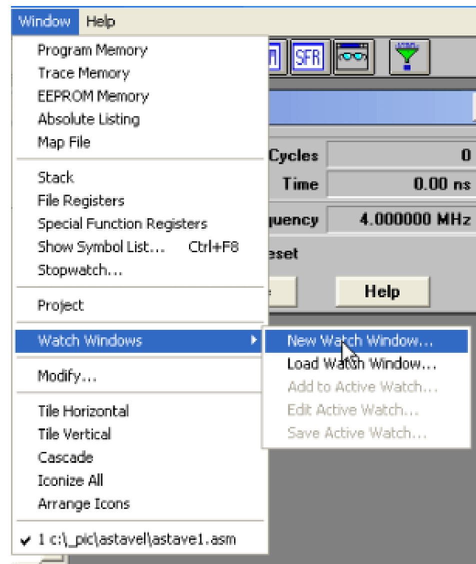
Para que a simulação possa ocorrer, é necessário que a janela que contenha o fonte esteja ativada. Se não estiver, basta dar um click dentro da janela. Use as teclas de simulação e observe... Tecle F6 para resetar.

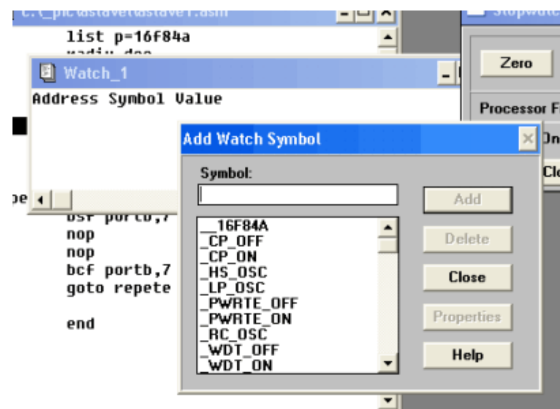
Tecle F7 para simular passo a passo ou Ctrl + F9 para animar. Pare a simulação com F5, zere o stopwatch, continue simulando e observe, ela marca os ciclos de máquina e o tempo decorrido desde que você zerou o stopwath. Quando Resetamos o stopwatch também é zerado.



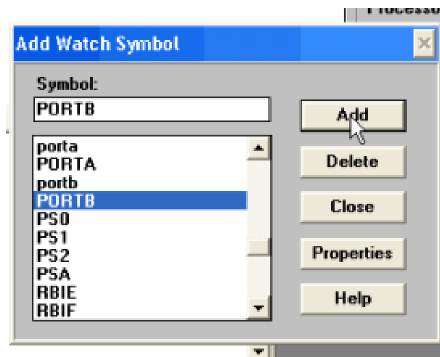
- Janela de Observação dos registros do PIC
  Podemos adicionar no nosso ambiente integrado mais uma janela para observar o que acontece com os registros do PIC durante a execução de cada instrução. Para facilitar o MPLAB já coloca os registros destinados às variáveis do seu programa, com os nomes que você definiu no fonte. Além disso você pode definir como ver os registros, se em decimal, hexadecimal, binário etc.

Clique em Window > Watch Windows > New Watch Window...

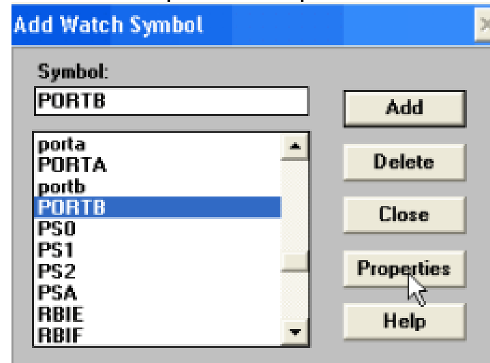




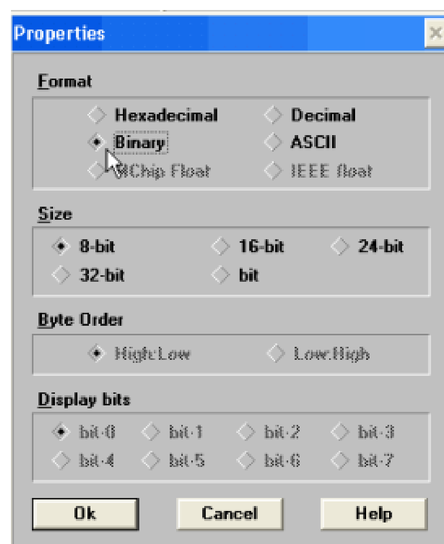
No campo Symbol escreva PORTB, ou procure esse nome na caixa de procura abaixo do campo symbol acionando a barra de rolagem e clicando no registro desejado.
Observação: os registros do PIC devem ser selecionados em maiúsculas



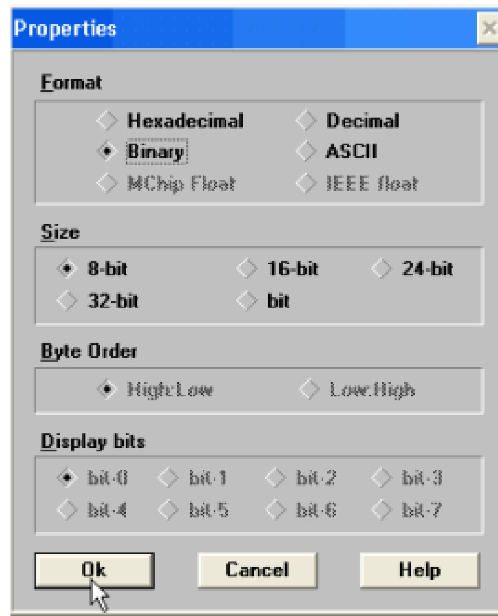
Cliqueem Properties



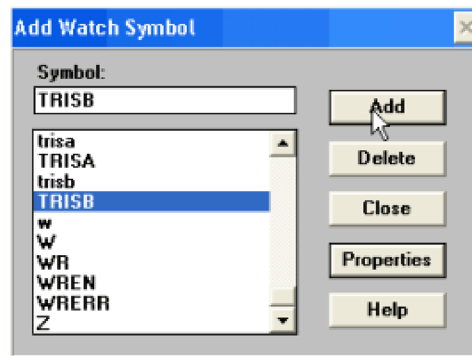
Defina em Format Binary e Size 8 bits isso vai mostrar na janela de observação o valor do portb em binário de 8 bits



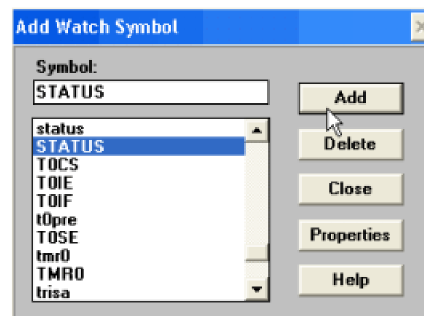
Clique em OK



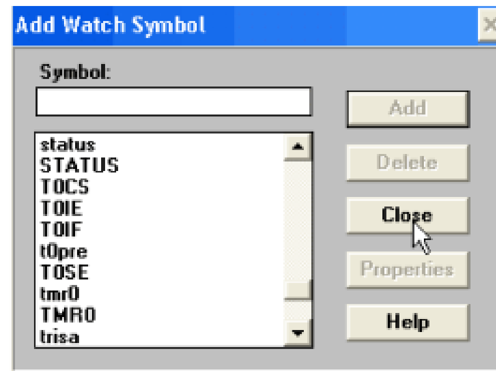
Selecione TRISB e clique em ADD, como já definimos no registro anterior o binário de 8 bits, ele mantém a última propriedade editada.



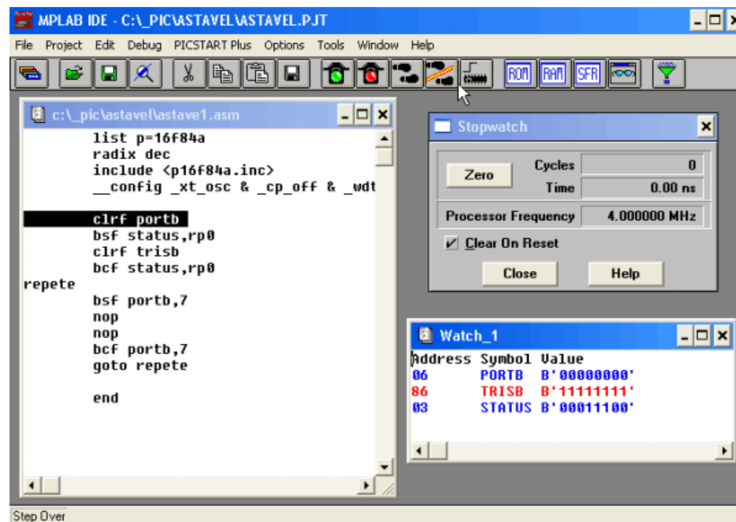
Selecione STATUS e clique em ADD



Agora feche clicando em Close

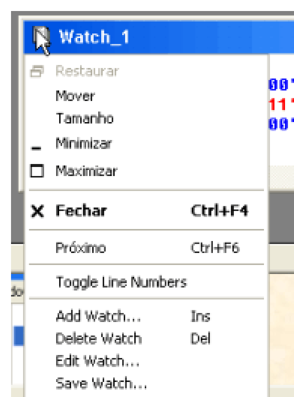


Arrume as janelas no seu Mplab para ver todas.

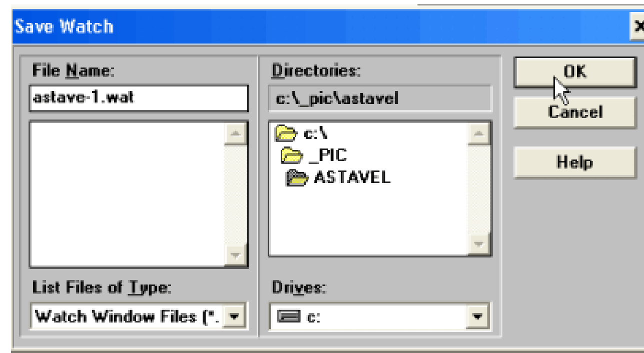


Lembre-se que a janela do fonte deve estar ativa para que o MPLAB possa fazer a simulação. Execute as teclas de simulação e observe o tempo decorrido e os registros do PIC. Use a simulação animada, vc poderá observar o bit 7 do portb alternando de zero pra um.

Clique com a lado direito do mouse sobre o ícone de um bloquinho de notas no Watch_1



Neste menu você encontra comandos para Adicionar outras janelas, Deletar uma janela, Editar a janela corrente ou salvar sua janela para uso posterior. Clique em salve watch



Salve a janela como astave_1.wat

Nós só vimos o básico até agora, existem outras simulações, por exemplo simular as entradas, mas isso no decorrer do curso a gente vai estudando.

Exercícios

1. Faça mais testes com o simulador, edite a janela, as propriedades de como o registro é apresentado: decimal hexa. Insira um outro registro portb com propriedades diferentes.
2. Faça um novo projeto seguindo o que aprendeu com o fonte abaixo, faça a compilação e a simulação com base na janela de registro descreva o que faz esse programa. mande-me a resposta pelo canal de comunicação. (acerte as tabulações) " nesse fonte tem uma instrução que ainda não vimos, mas com base na observação da janela de registro você pode deduzir"

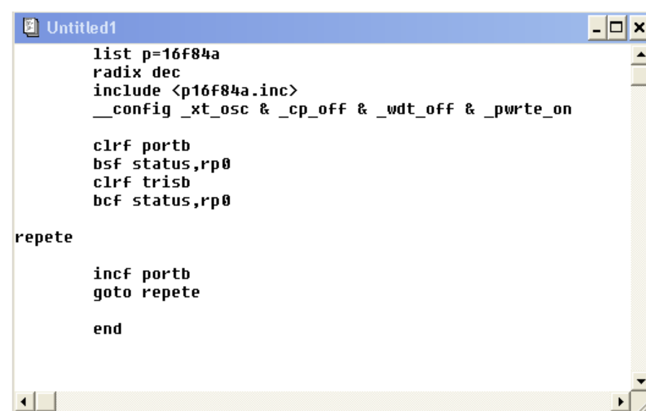
```
        list p=16f84a
        radix dec
        include <p16f84a.inc>
        __config _xt_osc & _cp_off & _wdt_off & _pwrte_on

        clrf portb
        bsf status,rp0
        clrf trisb
        bcf status,rp0

repete

incf portb
goto repete

end
```



Considerações sobre o MPlab

Note que até este momento do curso, me preocupei apenas com a ferramenta de desenvolvimento, o IDE MPlab, " Ambiente Integrado de Desenvolvimento MicrochiP LABoratório". O que vimos até agora é muito pouco sobre esta poderosa ferramenta, mas é com esse mínimo já temos condições de iniciar o estudo dos microcontroladores PIC, durante as aulas vamos discorrer mais alguns detalhes do MPlab, mas com certeza não iremos ver tudo sobre ele, acredito que iremos usar apenas uns 10% ou menos dos recursos disponibilizados, um curso completo do Mplab duraria pelo menos umas 100 horas. A grande maioria dos programadores são autodidatas, e o próprio software é bem intuitivo. Quando precisar de mais recursos, basta pegar o Guia do usuário, (Manual doMplab), e algumas HB que se acaba descobrindo as coisas.

HB significa "horas bunda", horas e horas sentado numa cadeira na frente do computador se dedicando ao estudo.

E cá entre nós, dominar a programação do pic exige centenas de HB. nosso curso é apenas o começo, o start para uma tecnologia que não tem fim, com certeza você terá sua maneira personalizada de programar, de resolver os problemas que encontrar. Mas contamos com uma grande ajuda, que é a própria Microchip, que disponibiliza toda a informação técnica de forma gratuita na internet.

Espero que nesse ponto do curso, você consiga sem ter que ficar olhando em suas anotações, a criar um projeto completo:

- Organizar os projetos em pastas separadas;
- Digitar um novo código fonte e salvar na pasta do projeto;
- Criar o projeto propriamente dito, editando as propriedades do arquivo alvo, o .HEX, adicionar o node do arquivo fonte .ASM;
- Compilar, descobrir os erros de sintaxe;
- Criar janelas de simulação, simular usando as teclas de atalho.

Quando achar que preenche esse quesito, você estará pronto para o próximo passo, a programação do PIC.

EXERCICIOS

Crie um projeto com os seguintes arquivos e faça testes

```
list            p=16f84a
        radix           dec
        include <p16f84a.inc>
        __config _xt_osc & _cp_off & _wdt_off & _pwrte_on

        clrf    portb
        clrf    porta
        bsf     status,rp0
        clrf    trisb
        clrf    trisa
        bcf     status,rp0
repete
        bsf     portb,7
        bcf     porta,4
        nop
        nop
        bcf     portb,7
        bsf     gorta,4

                                        goto    repeti
```

(b)

```
LIST	P=16F84A
	RADIX	DEC
	INCLUDE		<P16F84A.INC>
	__CONFIG _XT_OSC & _CP_OFF & _WDT_OFF & _PWRTE_ON


x	equ	0ch
tempo	equ	0dh
led1	equ	7


	bsf	status,rp0
	bcf	trisb,7
	bcf	status,rp0

loop:
	bcf	portb,led1
	call	ms100
	bsf	portb,led1
	call	ms100
	goto	loop



ms100
	movlw	100
	movwf	tempo
ms1
	movlw	249
	movwf	x
ms2
	nop
	decfsz	x
	goto	ms2


	decfsz	tempo
	goto	ms1
	return
	end
```

(c)

```
list		p=16f84a
	radix		dec
	include	<p16f84a.inc>
	__config _xt_osc & _cp_off & _wdt_off & _pwrte_on

	clrf	portb
	clrf	porta
```

```
        bsf     status,rp0
        clrf    trisb
        clrf    trisa
        bcf     status,rp0
repete
        incf    portb
        decf    porta

        goto    repete

        end
```

Descreva os problemas encontrados e como resolveu.

Microcontroladores

Tipicamente os microcontroladores se caracterizam por incorporarem internamente cpu, memórias de programa e dados e vários periféricos como timers, watchdog timers, comunicação serial, conversores analógicos digitais, geradores de PWM, etc. Fazendo com que a aplicação final fique extremamente compactada

Microchip

A Microchip é uma empresa norte americana, fundada em 1989, com sede na cidade de Chandler, Arizona (oeste dos E.U.A.) Esta empresa desenvolve, fabrica e comercializa microcontroladores (PIC), memórias seriais (I2C e SPI), produtos para segurança (Keeloq), identificadores por RF (RFID), conversores A/D, circuitos integrados de supervisão (Bronw out) e amplificadores operacionais. Principais Endereços:

Estados Unidos:

Corporate Headquarters Microchip Technology Inc
2355 West Chandler Blvd. Chandler, Arizona, USA 85224-6199

Brasil:

A Microchip é representada no Brasil pela empresa Artimar. Os micro-controladores PIC. podem ser comprados junto aos distribuidores autorizados: Aut-Comp, Future e Hitech.

Família dos Microcontroladores PIC

A Microchip é uma precursora no uso da tecnologia RISC em microprocessadores. O nome RISC é a abreviação de Reduced Instruction Set Computer (computador com conjunto de instruções reduzido).

Diferente da arquitetura Von Newmann, a estrutura RISC é baseada na arquitetura Harvard que possui um barramento para dados e outro para o programa, e tem como características, tamanhos diferenciados entre barramento de dados e de programa, permitindo que em uma única palavra, está a instrução e o operando. Existem modelos de PIC onde o barramento de dados é de 8 bits e o de programa é de 12 bits. Com isso conseguimos compactar o código e executa-lo em alta velocidade.

A microchip oferece várias famílias de microcontroladores de 8 bits, que se adaptam aos mais variados projetos. Entre elas podemos citar:

PIC 12C508 (microcontrolador de 8 pinos), 16F84 (microcontrolador de 18 pinos com memória flash, EEPROM, RAM, e muito mais), 16FXXX (com mais periféricos, como comparadores de tensão, conversor A/D, UART e outros) .

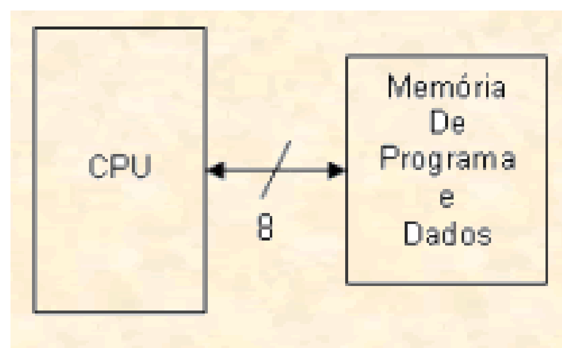
Característica da tecnologia RISC

O alto desempenho da família de microcontroladores PIC pode ser atribuída as seguintes características de arquitetura:

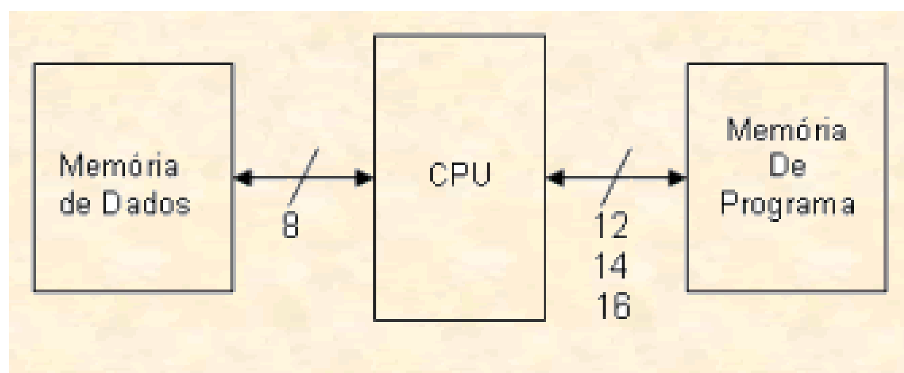
- Arquitetura Harvard
- Conceito de registrador arquivo
- Todas as instruções com palavras simples
- Palavra de instrução longa (LWI - Long Word Instruction)
- Arquitetura de instruções em "Pipeline"
- Instruções de apenas um ciclo de máquina
- Conjunto de instruções reduzido

Arquitetura Harvard x Von Newmann

Na arquitetura Von Newmann tradicional utiliza o mesmo barramento para memória de programa e dados.



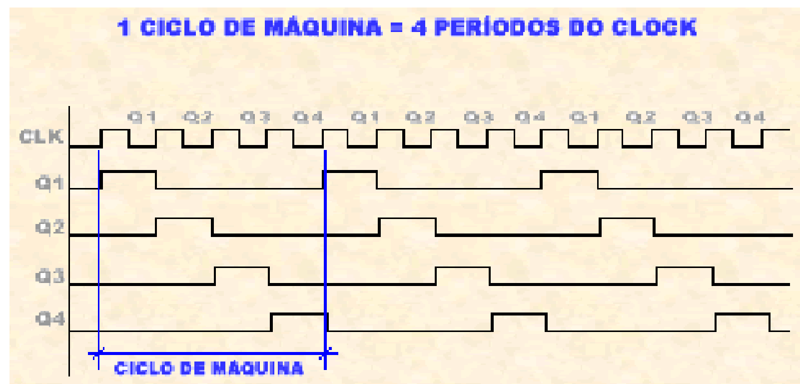
Na arquitetura Harvard utiliza um barramento para memória de programa e um para memória de dados



Ciclo de Instruções

A entrada de clock (pino OSC1 CLKIN) é internamente dividida por quatro para gerar quatro clocks em quadratura sem sobreposição, nomeados Q1, Q2, Q3, e Q4. Internamente o contador

de programa PC é incrementado em Q1, e a instrução é retirada da memória de programa e colocada no registrador de instruções em Q4. Ela é decodificada e executada durante o ciclo seguinte de Q1 até Q4.



Para calcular o tempo de cada ciclo de instrução realizado, baseado no dispositivo oscilador, por exemplo um cristal, basta fazer o seguinte cálculo:

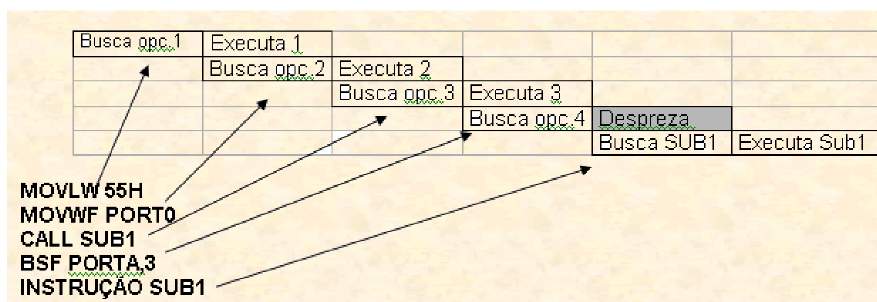
$$Tcy = \frac{1}{(fosc/4)}$$

Onde:

Tcy – Ciclo de Máquina
Fosc – Frequência do oscilador

Fluxo de Instrução/ Pipeline

Um ciclo de instrução consiste de quatro ciclos Q (Q1, Q2, Q3, Q4). A busca e execução são feitas em linha, de tal forma que a busca leva um ciclo de instrução e a execução leva outro ciclo. Contudo, devido à característica de "Pipeline", cada instrução é executada efetivamente em um ciclo, pois simultaneamente ocorrem as execuções de uma instrução e a busca a instrução seguinte. Se a instrução causa a alteração no contador de programa, então dois ciclos são necessários para completar a instrução.



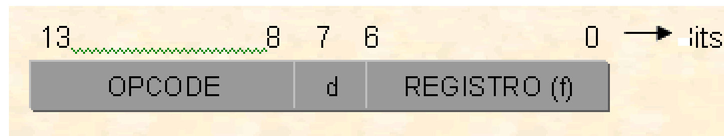
Palavra de Instrução Longa

A arquitetura com barramentos separados para instruções e dados permitem larguras de barramento diferentes. Com isso o barramento de instruções é otimizado para uma palavra de comprimento única. O número de bits do barramento de instruções depende de quantas instruções são implementadas e do número de registradores disponíveis em cada família de microcontrolador.

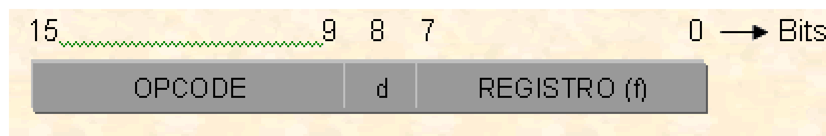
PIC 12C5XX - Instruções de 12 bits



PIC 16FXXX - Instruções de 14 bits



PIC 17CXX - Instruções de 16 bits



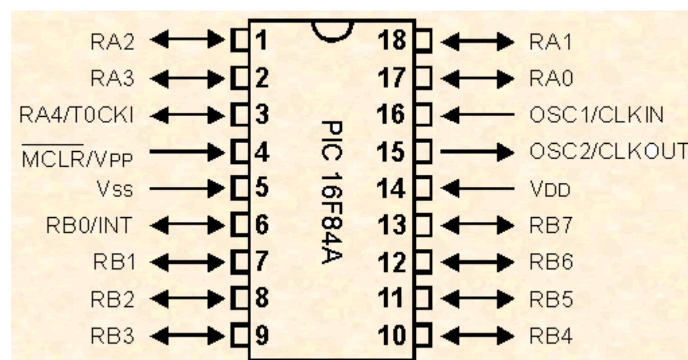
Microcontrolador PIC16F84

O PIC 16F84 é um microcontrolador que pode operar de DC até 10 MHz (ciclo de instrução de 400 ns) e devido as suas características de projeto funciona com o mínimo de componentes externos. O PIC 16F84A pode operar até 20 MHz.

Características principais

- 1 K (1024) palavras de 14 bits para programa;
- 68 bytes de RAM para uso geral;
- 64 bytes de EEPROM para dados;
- Stack com 8 níveis;
- Apenas 35 instruções;
- 15 registros específicos em RAM para controle do chip e seus periféricos;
- Timer de 8 bits com opção de prescaler de 8 bits;
- 13 pinos que podem ser configurados individualmente como entrada e saída;
- Alta capacidade de corrente nos pinos (podendo acender um led);
- Capacidade de gerenciar interrupções (até 5 entradas), do timer e EEPROM;
- Watch Dog para recuperação e reset em caso de travas no software;
- Memória de programa protegida contra cópias;
- Modo Sleep para economia de energia;
- Várias opções de osciladores.

Pinagem e características elétricas básicas



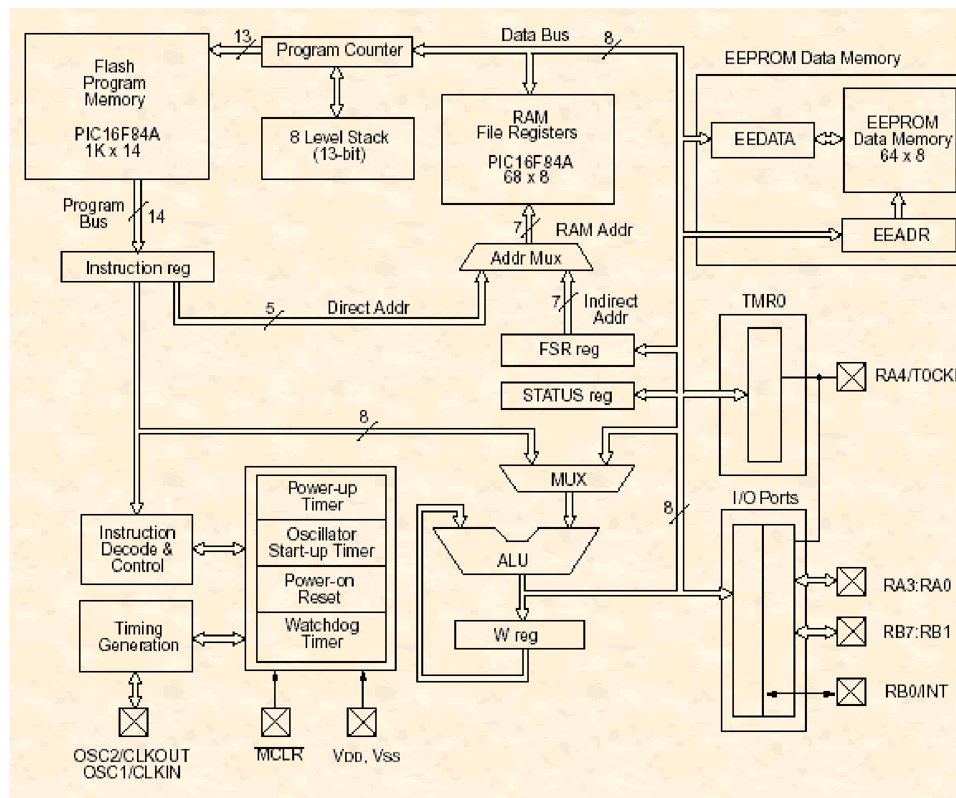
Faixa de Alimentação: 2 a 6 volts - típico 5 volts
Consumo de corrente: 1) < 2 mA a 5 volts a 4 MHz

2) 15 A a 2 volts a 32 KHz
3) 2 A a 2 volts em stand by

Descrição dos Pinos do 16F84

1. RA2 É um pino de I/O programável em entrada ou saída da unidade. Corresponde ao BIT 2 da PORTA A.
2. RA3 É um pino de I/O programável em entrada ou saída da unidade. Corresponde ao BIT 3 da PORTA A.
3. RA4 / RTCC ou T0CKI É um pino multi-função que pode ser programado como uma linha normal de I/O ou como linha de clock para entrada em sentido ao contador RTCC ou TMR0. Se programada como pino de I/O corresponde ao BIT 4 da PORTA A ao contrário de outra linha de I/O, Quando esta linha funciona como saída, trabalha em coletor aberto.
4. MCLR / VPP Em condição normal de funcionamento desenvolve a função de Master CLeaR ou seja Reset estará ativo a nível 0. Pode ser conectado a um circuito de reset externo ou simplesmente conectando-o ao positivo da alimentação. Quando o PIC vier posto em Program Mode será utilizado como entrada para a tensão de programação Vpp.
5. VSS É o pino que vai conectado ao negativo da tensão de alimentação.
6. RB0 É um pino de I/O programável em entrada ou em saída. Corresponde ao BIT 0 da PORTA B e pode ser programada para gerar interrupção.
7. RB1 É um pino de I/O programável em entrada ou em saída. Corresponde ao BIT 1 da PORTA B
8. RB2 É um pino de I/O programável em entrada ou em saída. Corresponde ao BIT 2 da PORTA B.
9. RB3 É um pino de I/O programável em entrada ou em saída. Corresponde ao BIT 3 da PORTA B.
10. RB4 É um pino de I/O programável em entrada ou em saída. Corresponde ao BIT 4 da PORTA B.
11. RB5 É um pino de I/O programável em entrada ou em saída. Corresponde ao BIT 5 da PORTA B.
12. RB6 É um pino de I/O programável em entrada ou saída. Corresponde ao BIT 6 da PORTA B.
13. RB7 É um pino de I/O programável em entrada ou saída. Corresponde ao BIT 7 da PORTA B.
14. VDD É o terminal positivo de alimentação do PIC. em todas as três versões disponíveis do PIC16F84 (comercial, industrial e automotiva) a tensão pode assumir um valor que vai de um mínimo de 2.0 volts a um Maximo de 6.0 volts.
15. OSC2 / CLKOUT É um pino de conexão no caso de se utilizar um cristal de quartzo para gerar o clock. E como saída de clock caso for aplicado um oscilador RC externo.
16. OSC1 / CLKIN É um pino de conexão para o caso de se utilizar um cristal de quartzo ou um circuito RC para gerar o clock. E também como entrada caso utilizemos um oscilador externo.
17. RA0 É um pino de I/O programável em entrada ou saída. Corresponde ao BIT 0 da PORTA A.
18. RA1 É um pino de I/O programável em entrada ou saída. Corresponde ao BIT 1 da PORTA A.

Arquitetura Interna do PIC 16F84



Os membros da família 16FXXX podem acessar tanto direta como indiretamente qualquer posição de memória RAM ou de registros internos, pois estão todos mapeados no mesmo bloco de memória.

Qualquer operação pode ser feita com qualquer registro (de dados ou de controle).

As operações lógicas e aritméticas são realizadas por um bloco chamado de ULA (unidade lógica e a aritmética) que possui um registro próprio chamado W (Working register - popular acumulador), Vamos usar muito esse registrador, que não está presente na RAM e não é acessado por endereçamento. A ULA é de 8 bits e permite realizar somas, subtrações, deslocamento (shifts) e operações lógicas.

Os bits de sinalização, ou flags, chamados Z (zero), C (carry) e DC (digit carry) refletem os resultados de várias operações realizadas na ULA, e ficam armazenados no registrador STATUS.

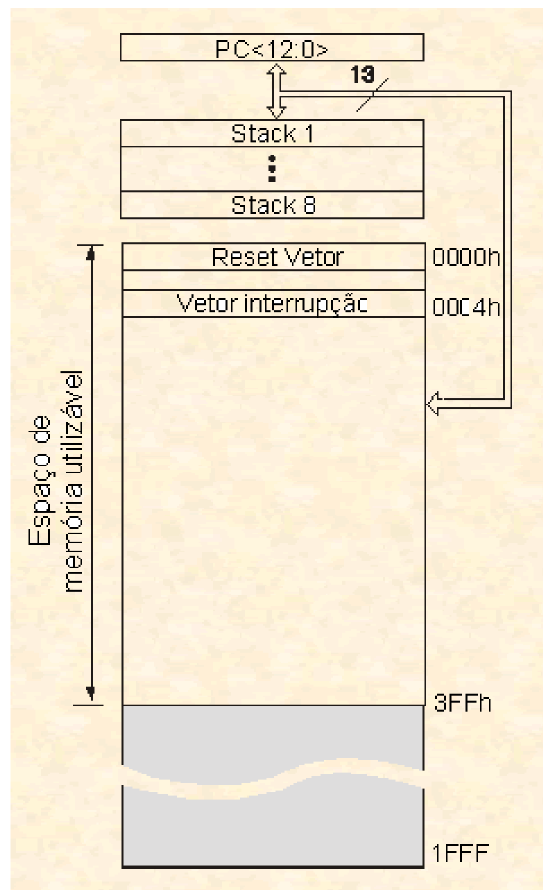
Organização da Memória

Podemos notar que o PIC 16F84 tem duas memórias principais, a de *dados* e a de *programa*. A de *programa* é onde armazena o programa, a sequência de instruções que foi convertida em hexadecimal pelo compilador, que irá gerenciar o pic. A de *dados* é para armazenamento temporário, ambas possui uma organização que devemos conhecer.

Organização da Memória de programa

A memória de programa varia em tamanho e organização. Nos membros da família 16FXX a memória de programa é dividida em páginas, isso se deve à limitação de endereçamento direto dado pelo contador de programa (PC) que tem 13.
O PIC 16F84 possui apenas 1K implementado (de 00 a 3FF). Qualquer referência a outras posições de memória será "deslocada" para este bloco de 1K.
Exemplo: As posições 72h ,472h, C72h e outras somadas 400H referem-se sempre a posição original 72h.

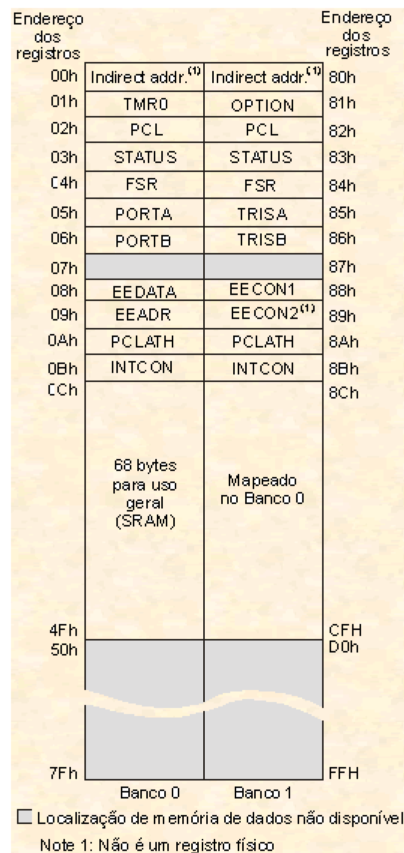
Mapa de Memória de Programa



- Os Stack (pilha) que tem um espaço reservado que não faz parte da memória utilizável pelo usuário.
- O espaço utilizável que vai de 0000h a 3FFh (1024 posições de 14 bits).
- Vetor de reset (000h), que é a primeira posição que o PC aponta, quando o PIC é resetado.
- Vetor de interrupção, que ao receber um pedido de interrupção externa, o PC aponta para o endereço 0004h.

Mapa de Memória de Dados e Registro de Controle

A memória de dados e memória de registro de controle nada mais são que um grupo de memória RAM, organizadas em dois bancos de registradores: banco 0 e 1.



Os Registros especiais e a memória de dados estão organizadas conforme a figura acima. Temos o banco 0 e o banco 1, que serão selecionados através de dois flags, 2 bits, RP0 e RP1, podendo selecionar até quatro bancos. Como o 16F84 possui apenas dois bancos, o RP1 ficará sempre em 0.

Vale salientar que por ser uma memória RAM, ao desligar a alimentação, os dados nela gravado serão perdidos.

Memória de Uso Geral

A memória de uso geral se estende do endereço 0Ch a 4Fh (no banco 0), totalizando 68 bytes disponíveis ao usuário. No banco 1 de 8Ch a CFh está na verdade mapeado no banco 0, portanto qualquer endereço no banco 0 está espelhado no banco 1, ou seja, se eu acessar o endereço 0Ch é o mesmo que acessar 8Ch.

Esta memória será utilizada para alocar variáveis, bem como, salvar informações úteis quando houver chamada de sub-rotina ou pedidos de interrupção.

Arquivos de Registros Especiais (SFR)

Os SFR ou melhor dizendo Registros de Controle ocupam posições de memória RAM que vai do endereço 00h ao 0Bh no banco 0, e de 80h ao 8Bh no banco 1, onde cada posição com seu respectivo endereço recebe um nome. Alguns registros se repetem no banco 0 e 1, podendo ser programado tanto em como no outro, como por exemplo o registro Status. Tanto os registros como a memória de dados de uso geral são de 8 bits.

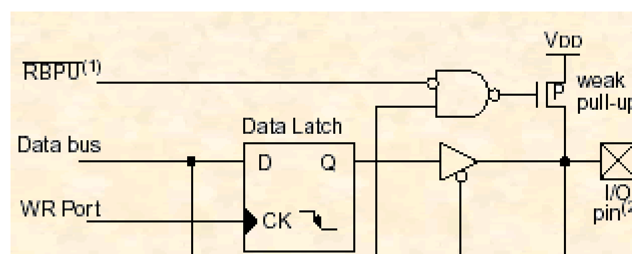
A maioria destes registros podem ser programados bit a bit, são através deles que teremos o controle geral do PIC.

Temos a seguir um resumo para que serve cada registro, pois veremos com mais detalhes durante as próximas aulas.

- INDF - Endereçamento indireto
- TMR0 - Registro de contagem do timer 0
- PCL - Parte baixa do contador de programa
- STATUS - Registro status para controle da CPU
- FSR - ponteiro para o endereçamento indireto
- PORTA - Registro dos pinos do PORTA
- PORTB - Registro dos pinos do PORTB
- - - Não implementado
- EEDATA - Dado lido/gravado na EEPROM
- EEADR - Endereço para ler/gravar na EEPROM
- PCLATH - Parte alto contador de programa
- INTCON - Registro INTCON para controle da CPU
- OPTION - Registro OPTION para controle da CPU
- TRISA - Direção dos pinos do PORTA
- TRISB - Direção dos pinos do PORTB
- EECON1 - Controle da EEPROM
- EECON2 - Controle da EEPROM
-

Registros de Controle da CPU

Os Microcontroladores, possuem resgistros, ou files, ou ainda palavras de memórias, que são responsáveis pelo controle do circuito interno do PIC. veja esse exemplo que esclarece bem o que é um controlar um hardware por software:



O circuito acima faz part de um pino do pic, que pode ser entrada ou sáida, o /RBPU, quando coloco barra na frente do nome é por que o bit é negado, ou seja ativa em zero, RBPU significa RB é o nome de um pino do pic que é controlado pelo portb, PU é pull up, então esse bit ativa um resistor de pull up interno ao pic nesse pino, isso facilita o circuito externo, economizando uma resistência. Como funciona? Quando o pino é entrada a entrada de baixo da porta nand é 1, então se /RBPU é 0 a saida da nand fecha o transistor de efeito de campo e há o pull up, se /RBPU é 1 o transistor fica cortado e não há pull up. O bit /RBPU é o sétimo do registro chamado OPTION, vc vai ver mais à frente.

Existem três registros importantes para controle da CPU: STATUS, OPTION e INTCON, além dos registros das portas e outros. Por estes registros que teremos controle sobre flags da ULA, interrupção, timers e outros.

Vamos nesse momento fazer um breve comentário sobre os bits dos registros, mas vai ser nas aplicações práticas que vamos estudar e entender melhor a função de cada um. por isso se não entender 100% o que eles são, não se preocupe, nas aplicações fica esclarecido. o importante aqui nessa aula, é saber que no pic existem registros, ou files especiais que controlam todo o pic, o qual a maioria temos pleno acesso de leitura e escrita.

Para descrevermos os registros, que a Microchip também chama de files, usaremos a seguinte nomenclatura:

Propriedade	*R* - podemos ler o bit
	W - podemos escrever o bit, ou seja, podemos alterara-lo
Situação do bit no Reset	0 - bit em zero, ou clear
	1 - bit em um, ou set
	U - bit tem seus valores inalterados
	X - bit com valor indeterminado
	S - bit apenas "setavel", vc só pode setar o bit, sómente o hardware pode colocá-lo em zero.
Bit	Número do bit dentro do registro, de 0 a 7
Nome	Mnemônico do nome do bit, refere-se ao que está associado

Registro STATUS

O registro STATUS, configura os bancos de registros, flags da ULA e outros. Seu endereço físico é 03h (banco 0) e 83 (banco1). Valor no reset: 00011XXX

Propriedade	R/W	R/W	R/W	R	R	R/W	R/W	R/W
Reset	0	0	0	1	1	X	X	X
Bit	7	6	5	4	3	2	1	0
Nome	IRP	RP1	RP0	/T0	/PD	Z	DC	C

IRP - Seleciona bancos (para endereçamento indireto), é usado como o nono bit de um registrador de endereço, está no 16f84, mas aconselha-se não usá-lo, está aí porque faz parte dos pic´s com mais memória e periféricos.O IRP não é usado pelo 16F84A, devendo ficar em 0.
0 = 0,1 (00h - FFh)
1 = 2,3 (100h - 1FFh)

RP1 e RP0 - Seleciona os bancos de memória no endereçamento direto. Cada banco tem 128 bytes.		
RP1	RP0	Banco selecionado
0	0	Banco 0 (00h - 7Fh)
0	1	Banco 1 (80h - FFh)
1	0	Não utilizado pelo 16F84
1	1	Não utilizado pelo 16F84

TO Bit sinalizador do Timer-out	
1	Após power-up, instrução CLRWDT ou SLEEP. Power-up significa que o pic está ativo, ligado, executando o programa. o CLRWDT, é CleaR WaTch Dog, o Watch Dog ou simplesmente o wdt é um temporizador "cão de

	guarda" que quando habilitado deve ser periodicamente resetado, e se isso não ocorrer é porque o programa "travou", ou deu algum problema sério, então o wdt reseta o pic. O Sleep é uma instrução que coloca o pic em standby economizando energia, um exemplo disso é um controle remoto, que só deve enviar um sinal se um botão for pressionado, caso não tenha botão pressionado o pic fica em stantby.
0	Ocorreu o timer-out do Watch Dog

\/PD Bit Power-down	
1	Após o power-up ou pela instrução CLRWDT
0	Pela execução do SLEEP

Z - Bit sinalizador de zero	
1	O resultado de uma operação lógica ou aritmética deu zero, isto é o registrador W é 00h
0	O resultado de uma operação lógica ou aritmética não deu zero, isto é o registrador W não é 00h

DC - Digit Carry/Borrow	
1	Ocorreu um carry-out do 3º para o 4º bit do W, numa operação de adição
0	não ocorreu um carry-out

C - Carry/Borrow	
1	Ocorreu um carry-out do 7º bit do resultado em W, numa operação de adição
0	Não ocorreu um carry-out

Situação após reset: Banco de memória em 0, bits sinalizadores de time-out e power-down setados, bits da ULA indeterminados.

Registro OPTION

O registro option configura o prescaler de temporização, timers e outros. Seu endereço físico é 81h. Valor no Reset: 11111111.

Propriedade	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1
Bit	7	6	5	4	3	2	1	0
Nome	/RBPU	INTDG	T0CS	T0SE	PSA	PS2	PS1	PS0

/RBPU bit de habilitação de Pull-up
1 - PORTB Pull-ups desabilitados
0 - PORTB Pull-ups habilitado

INTEDG - bit seleciona como aceitará a interrupção
1 - Na subida do sinal no pino RB0/INT
0 - Na descida do sinal no pino RB0/INT

T0CS - bit de seleção de fonte de clock do timer 0
1 - Transição no pino RA4/T0CKI
0 - Clock interno (CLKOUT = Fosc/4)

T0SE - bit de seleção de como incrementará o Timer 0
1 - Na descida do sinal no pino RA4/T0CKI
0 - Na subida do sinal no pino RA4/T0CKI

PSA - Bit de atribuição do Prescaler
1 - Prescaler atribuído ao Watch Dog
0 - Prescaler atribuído ao TMR0

PS2, PS1 e PS0 - Ajustam a taxa de divisão do Prescaler Veja a tabela na página seguinte.				
PS2	PS1	PSO	Divisão Timer 0	Divisão Watch Dog
0	0	0	1/2	1/1
0	0	1	1/4	1/2
0	1	0	1/8	1/4
0	1	1	1/16	1/8
1	0	0	1/32	1/16
1	0	1	1/64	1/32
1	1	0	1/128	1/64
1	1	1	1/256	1/128

Registro INTCON

O registro Intcon é para leitura e escrita, no qual se habilita bits para selecionar todos os tipos de interrupção. Seu endereço físico é 0Bh e 8Bh. Valor no Reset: 0000000X.

Propriedade	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	X
Bit	7	6	5	4	3	2	1	0
Nome	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

GIE - Global Interrupt Enable (bit de interrupção global)
1 - Habilita todas as interrupções desde que individualmente selecionadas.
0 - Desabilita todas as interrupções

EEIE - Bit de habilitação de interrupção no fim da escrita na EEPROM
1 - Habilita interrupção
0 - Desabilita interrupção

T0IE - bit para habilitar interrupção gerada pelo overflow no TMR0
1 - Interrupção habilitada
0 - Interrupção desabilitada

INTE - Bit para habilitar interrupção externa em RB0/INT
1 - Interrupção habilitada
0 - interrupção desabilitada

RBIE - Bit para Interrupção por mudanças no PORTB
1 - Interrupção habilitada
0 - Interrupção desabilitada

T0IF - Bit Sinaliza interrupção pelo Overflow do TMR0
1 - Ocorreu overflow no TMR0
0 - Não ocorreu overflow

INTF - Bit para sinalizar interrupção externa no pino RB0/INT
1 - Ocorreu pedido de interrupção

0 - Não ocorreu pedido de interrupção

RBIF - Bit para sinalizar interrupção por mudanças no PORTB
1 - Um ou mais pinos de RB4 a RB7 mudou de estado
0 - nenhum mudou de estado

PCL e PCLATH

Os registros PCL e PCLATH armazenam o endereço da linha de programa que será executada no momento (contador de programa - PC), podendo ser lido ou escrito. O PCL armazena os 8 bits menos significativos do PC, enquanto o PCLATH armazena os 5 bits mais significativos, formando um número de 13 bits. Isso se deve, por que a memória dos registros só armazenam 8 bits.

STACK

O stack permite armazenar uma combinação de 8 chamadas (call) de sub-rotinas e interrupções. O stack tem 8 níveis de profundidade de 13 bits, que tem a finalidade de armazenar o valor atual do contador de programa (PC) PC+1, quando ocorrer alguma chamada ou interrupção, isto permite o retorno ao endereço do programa principal após execução destas. O cuidado que deve se tomar, é não passar de 8 chamadas consecutivas sem retorno.

INDF - Endereçamento Indireto

O INDF (00h na RAM), não é na verdade um registro fisicamente implementado. Quando se acessa o INDF estamos na verdade acessando a posição indicada pelo registro FSR (File Selection Register - endereço 04h), que atua como um ponteiro para outras posições de memória.

Exemplo:
O registro 05h está com valor 10h
O registro 06h está com valor 0Ah
Armazena o valor 05h no FSR.
A leitura do INDF será o retorno do valor 10h
Incrementa em 1 FSR.
A leitura agora do INDF será o retorno do valor 0Ah

Portas de I/O

A maioria dos pinos das portas de I/O do PIC 16F84, são multiplexados com outra função, podendo ser alterada sua característica conforme o periférico ligado a sua entrada. Sendo assim nem sempre o pino de I/O é utilizado para este propósito.

No PIC 16F84, existem 13 portas de I/O, dividida em 2 portas distintas: PORTA com 5 bits (pinos) e PORTB com 8 bits (pinos).

PORTA e TRISA

O PORTA tem 5 pinos de I/O independentes chamados:

è RA0 - bit 0, saída com nível TTL
è RA1 - bit 1, saída com nível TTL
è RA2 - bit 2, saída com nível TTL
è RA3 - bit 3, saída com nível TTL
è RA4 - bit 4, divide função com T0CKI (entrada de timer externo), entrada Schmmitt Trigger, e saída dreno aberto.

Diagrama em blocos dos pinos RA0 a RA3

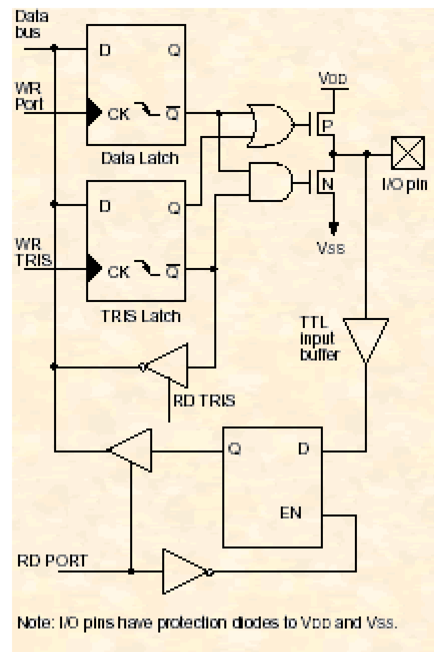
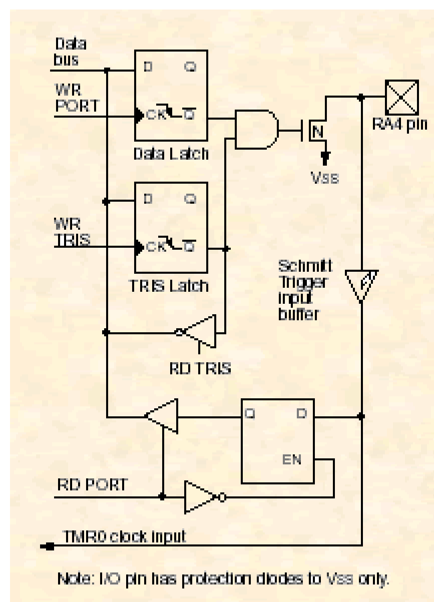


Diagrama em blocos do pino RA4



O PORTA, não lemos porta, mas sim "port a", só que se escreve tudo junto, é controlado pelo registro TRISA que determinará como irá trabalhar os pinos de I/O desta porta, ou seja, se o pino será entrada ou saída.

Propriedade	-	-	-	R/W	R/W	R/W	R/W	R/W
Reset	-	-	-	1	1	1	1	1

Bit	7	6	5	4	3	2	1	0
Nome	-	-	-	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0

TRISA4 a TRISA0 - Programa se cada pino do PORTA será entrada ou saída	
1	O pino está configurado para entrada (1 é parecido com I de imput)
0	O pino está configurado para saída (0 é parecido com O de output)

O endereço do registro PORTA é 05h, (banco 0), e o endereço do registro TRISA é 85h, (banco 1).

Reset por Power-on (ao ligar) : PORTA = - - - XXXXX, TRISA = - - - 11111
Demais resets : PORTA = - - - UUUUU, TRISA = - - - 11111

PORTB e TRISB

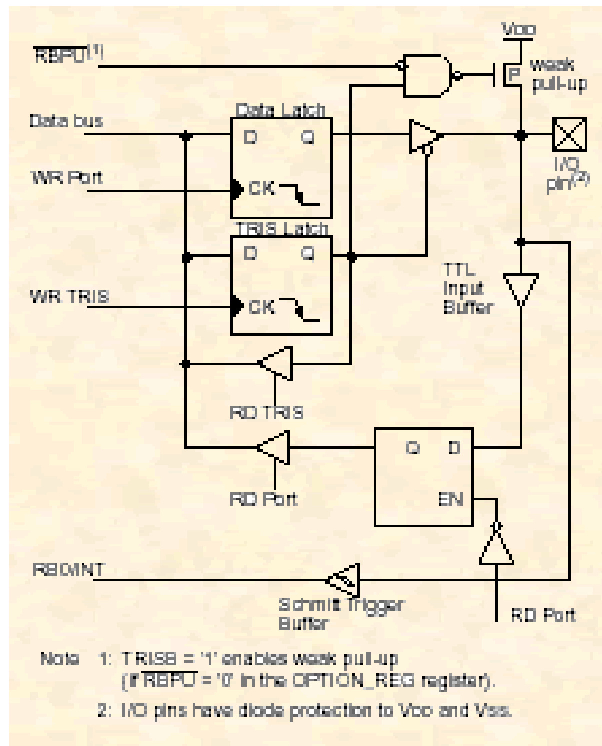
O PORTB tem 8 pinos de I/O independentes, o endereço de registro é 06h, e são chamados:

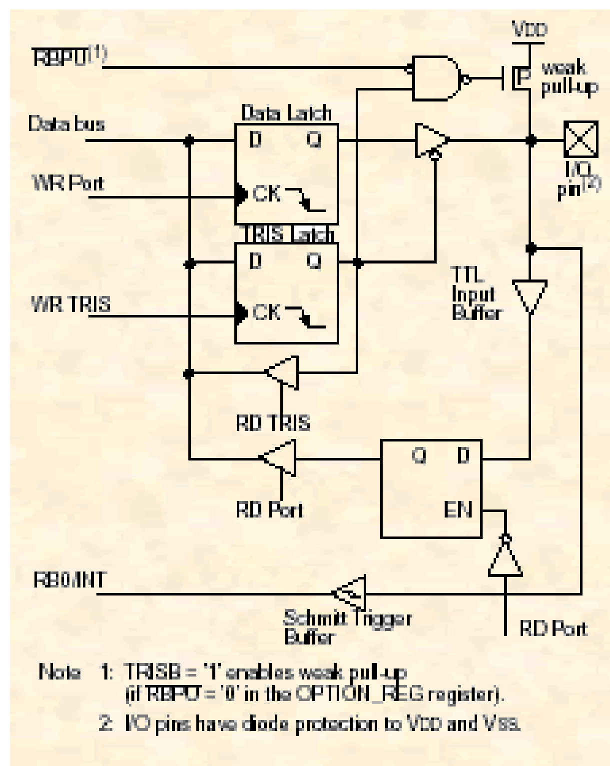
RB0 - bit 0, divide função com INT (interrupção externa), Saída com nível TTL/ST(1).
RB1 - bit 1, Saída com nível TTL
RB2 - bit 2, Saída com nível TTL
RB3 - bit 3, Saída com nível TTL
RB4 - bit 4, Saída com nível TTL, com interrupção por mudança de estado(3)
RB5 - bit 5, Saída com nível TTL, com interrupção por mudança de estado(3)
RB6 - bit 6, Saída com nível TTL/ST(2), com interrupção por mudança de estado(3)
RB7 - bit 7, Saída com nível TTL/ST(2), com interrupção por mudança de estado(3)

1 - Schmitt Trigger, quando configurado para entrada de interrupção externa
2 - Schmitt Trigger, quando em modo de programação serial
3 - Quando a interrupção por mudança de estado estiver configurada.

OBS: Todos os pinos podem ser configurados com PULL-UP interno, através do bit /RBPU, se o pino estiver configurado como entrada TRISB="1".

Diagrama em blocos dos pinos RB0 a RB3





Obs: Todos os pinos de I/O do PIC 16F84 tem diodo de proteção para VDD e VSS.

O PORTB é controlado pelo registro TRISB, que determinará se os pinos funcionarão como entrada ou saída. Seu endereço físico é 86h.

Propriedade	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1
Bit	7	6	5	4	3	2	1	0
Nome	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0

TRISB7 a TRISB0 - bits de Controle de direcionamento das portas	
1	O pino está configurado para entrada (1 é parecido com I de imput)
0	O pino está configurado para saída (0 é parecido com O de output)

Reset por Power-on (ao ligar) : PORTA = XXXXXXXX, TRISA = 11111111
Demais resets : PORTA = UUUUUUUU, TRISA = 11111111

Memória EEPROM de Dados

A memória EEPROM de dados pode ser lida e escrita durante a operação normal (com a tensão normal de alimentação). Esta memória não é diretamente mapeada no banco de registros, devendo ser endereçada através dos registros de funções especiais, sendo necessário quatro FSR para leitura e escrita em EEPROM. São eles: EECON1, EECON2 (registro não está implementado fisicamente), EEDATA e EEADR.

No EEDATA, armazenam os 8 bits (byte) para leitura ou escrita. No EEADR armazena o endereço da EEPROM que será acessado. No PIC 16F84 a EEPROM de dados tem um tamanho de 64 bytes, e seu endereço vai de 00h a 3Fh.

A escrita na EEPROM automaticamente grava sobre o dado armazenado anteriormente. A vantagem desta memória é que ao ser gravado uma informação, ela não se perderá ao desligar o sistema. Para uma escrita na EEPROM, gasta-se aproximadamente 10 mS, fator que "atrasa" o sistema, mas para leitura gasta o mesmo que uma leitura na RAM (se o clock for 4 MHz gastará 1 S).

Registro EECON1

O registro EECON1 é para controle das operações com a EEPROM. Seu endereço é 88h. Valor após o reset: UUU0X00X. O U será lido como 0.

Propriedade	U	U	U	R/W	R/W	R/W	R/S	R/S
Reset	-	-	-	0	X	0	0	0
Bit	7	6	5	4	3	2	1	0
Nome	-	-	-	EEIF	WRERR	WREN	WR	RD

EEIF - Bit sinalizador de interrupção de fim de escrita
1 - Já acabou a escrita (zerado por software)
0 - não acabou de escrever

WRERR - bit sinalizador de erro ao escrever na EEPROM
1 - Escrita prematuramente interrompida (por reset ou Watch Dog)
0 - Operação de escrita completada

WREN - Bit de habilitação de escrita na EEPROM
1 - Permite o ciclo de escrita
0 - Inibe a escrita de dados na EEPROM

WR - Bit de controle de escrita
1 - Inicia o ciclo de escrita. Será zerado por hardware assim que a escrita for completada
0 - A escrita na EEPROM foi completada

RD - Bit de controle de Leitura na EEPROM
1 - Inicia uma leitura na EEPROM. (é zerado por hardware. Gasta um ciclo)
0 - Não inicia leitura na EEPROM

Os bits RD e WR podem ser lidos, mas por software só pode setar.

Para iniciar uma operação de leitura ou escrita, basta colocar os valores em EEADR e EEDATA (na escrita) e setar os bits RD ou WR conforme a operação desejada.

Fusíveis de Configuração

Estes fusíveis ou bits de configuração podem ser programados quando ler 0 ou desprogramado quando ler 1, servindo para selecionar diversas configurações de dispositivos. Este espaço de memória fica localizado no endereço 2007h da memória de programa. Este endereço está distante da memória de programa do usuário e pertence a um espaço de memória especial para verificação/configuração (2000 - 3FFFh). Este espaço pode ser acessado apenas durante a programação.

Bit	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Nome	CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	PWRTE	WDTE	FOSC1	FOSC0
Endereço 2007h														

CP - do bit 13:4, proteção de código
1 - Código desprotegido
0 - Código PROTEGIDO

PWRTE - bit para habilitação do "power-up Timer

1 - Power-up Timer desabilitado
0 - Power up Timer habilitado

WDTE - bit para habilitação do Watch-Dog
1 - WDT habilitado
0 - WDT desabilitado

FOSC1, FOSC0 - Seleciona o tipo de oscilador		
FOSC1	FOSC0	Tipo de Oscilador
0	0	Cristal de baixa potência LP
0	1	Cristal ou ressonador de baixa velocidade XT
1	0	Cristal ou ressonador de alta velocidade HS
1	1	Modo RC externo

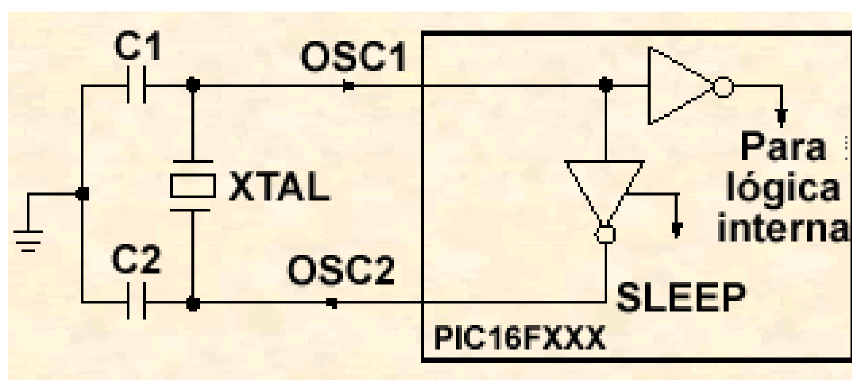
Configuração do Oscilador

O PIC 16F84A pode operar com quatro diferentes modos de osciladores, sendo selecionado através dos bits FOSC1 e FOSC0 dos fusíveis de configuração. Pode ser um dos modos descrito abaixo.

LP - Cristal de baixa potência
XT - Cristal ou ressonador de baixa velocidade
HS - Cristal ou ressonador de alta velocidade
RC - Resistor / Capacitor modo externo

Operação com Cristal/ressonador cerâmico

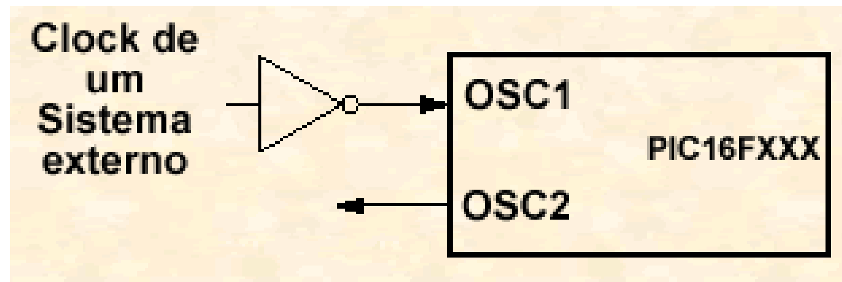
Qualquer cristal ou ressonador cerâmico selecionado em modo XT, LP ou HS será conectado nos pinos OSC1/CLKIN e OSC2/CLKOUT para estabelecer a oscilação. Veja a figura a seguir.



Os valores para C1 e C2 devem ser de 15 a 33 pF para um oscilador acima de 2 MHz.

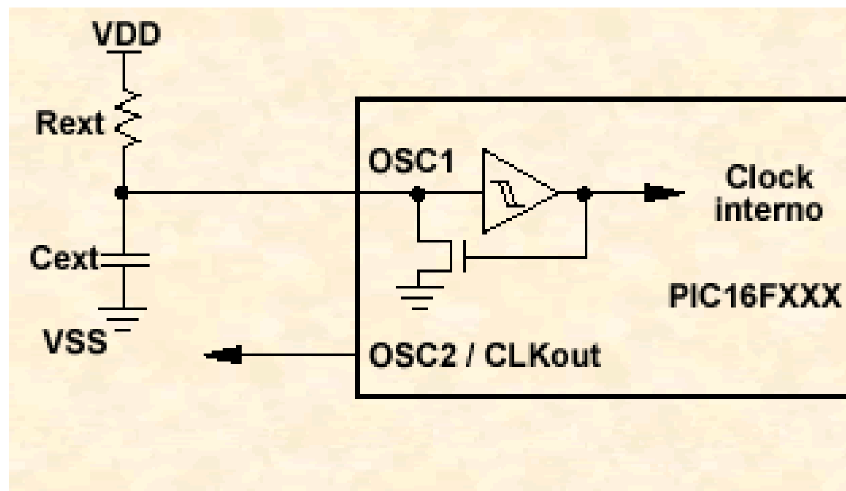
Clock de um Sistema Externo

O clock, operando com um oscilador externo, nos modos XT, HS ou LP, deve seguir o seguinte esquema de ligação:



Oscilador RC

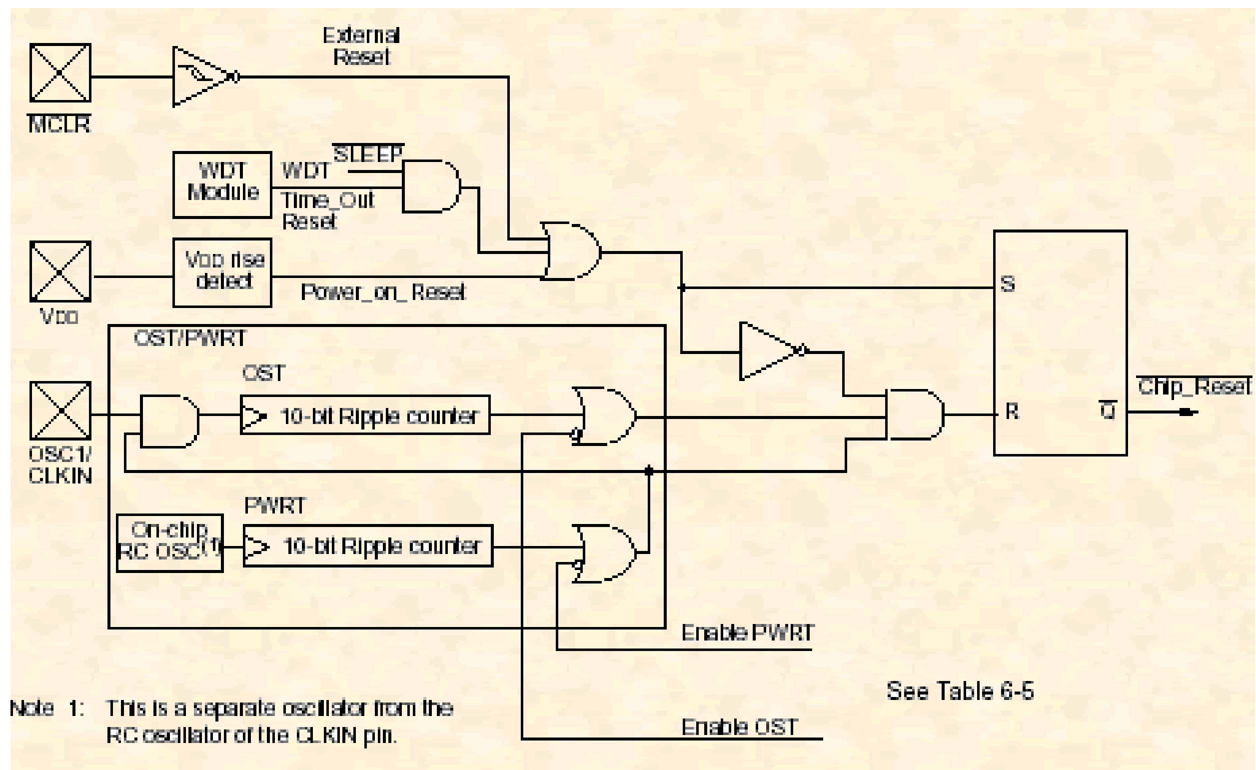
Para circuitos menos sensíveis a precisão na temporização, o modo RC oferece uma opção de custo menor. Depende apenas da alimentação, de um resistor e capacitor externo. Porém ficará susceptível a variações das características dos componentes (Rext e Cext), temperatura e valores dos componentes. Abaixo temos o esquema de ligação.



Recomenda-se valores: Para Rext - 5 K a 100 K, para Cext > 20 pF

Reset

Diagrama em blocos do circuito de reset do PIC, o reset é importante porque altera os valores dos registros quando é ativado, é muito importante sabermos disso.



Situação de todos os registros no reset

Pagina 24 do datasheet do 16f84A

Register	Address	Power-on Reset	$\overline{MCLR}$ Reset during: – normal operation – SLEEP WDT Reset during normal operation	Wake-up from SLEEP: – through interrupt – through WDT Time-out
W	—	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF	00h	---- ----	---- ----	---- ----
TMR0	01h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	02h	0000h	0000h	PC + 1 [2]
STATUS	03h	0001 1xxx	000q quuu [3]	uuuq quuu [3]
FSR	04h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA [4]	05h	---x xxxx	---u uuuu	---u uuuu
PORTB [5]	06h	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEDATA	08h	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEADR	09h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCLATH	0Ah	---0 0000	---0 0000	---u uuuu
INTCON	0Bh	0000 000x	0000 000u	uuuu uuuu [1]
INDF	80h	---- ----	---- ----	---- ----
OPTION_REG	81h	1111 1111	1111 1111	uuuu uuuu
PCL	82h	0000h	0000h	PC + 1
STATUS	83h	0001 1xxx	000q quuu [3]	uuuq quuu [3]
FSR	84h	xxxx xxxx	uuuu uuuu	uuuu uuuu
TRISA	85h	---1 1111	---1 1111	---u uuuu
TRISB	86h	1111 1111	1111 1111	uuuu uuuu
EECON1	88h	---0 x000	---0 q000	---0 uuuu
EECON2	89h	---- ----	---- ----	---- ----
PCLATH	8Ah	---0 0000	---0 0000	---u uuuu
INTCON	8Bh	0000 000x	0000 000u	uuuu uuuu [1]

Legend: u = unchanged, x = unknown, - = unimplemented bit read as '0', q = value depends on condition.
Note 1: One or more bits in INTCON will be affected (to cause wake-up).
2: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).
3: Table 6-3 lists the reset value for each specific condition.
4: On any device reset, these pins are configured as inputs.
5: This is the value that will be in the port output latch.

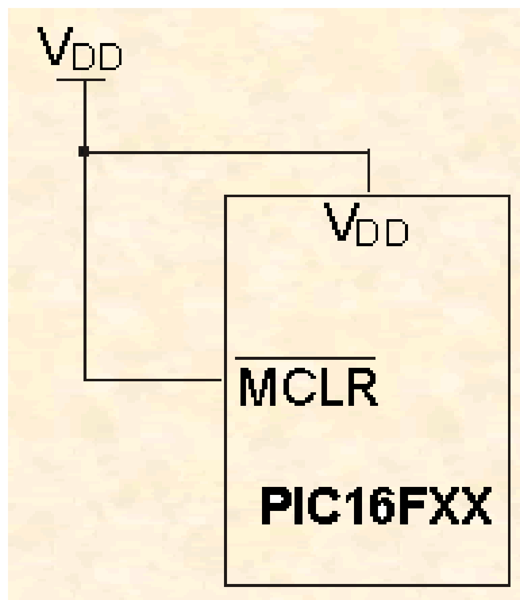
No PIC 16F84 existem vários tipos de reset:

Power-on Reset (POR)
reset, durante a operação normal
reset, durante o Sleep
WDT reset, durante a operação normal
WDT Wake-up, durante o Sleep

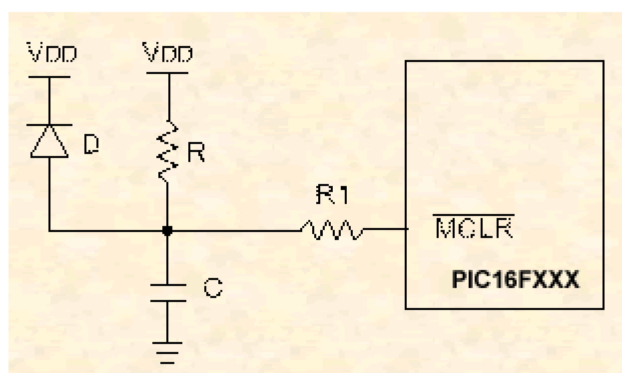
Alguns registros não são afetados em qualquer condição de reset. Seus estados são desconhecidos no reset POR e não mudam em qualquer outro reset.

Power On Reset (POR)

Um power on reset é um pulso gerado no chip, quando a elevação do VDD é detectada (numa faixa de 1,2 - 1,7 V). O POR é garantido somente se ligarmos a entrada do ao VDD, ou através de um resistor. Abaixo temos duas opções de ligação do POR.



Este primeiro circuito é a forma mais simples de gerar um reset no PIC, dispensando componentes externos com resistores e capacitores.



Já o segundo é o tradicional, fazendo com que o reset se prolongue por mais tempo. O diodo D, ajuda o capacitor se descarregar mais rápido quando a alimentação for desligada; Recomenda-se um resistor R menor que 40 K . E o resistor R1 é recomendado de 100 a 1 K para limitar a corrente proveniente do capacitor.

Power-up Timer (PWRT)

O power-up timer gera um tempo fixo a mais de 72 ms após o POR. O PWRT timer opera com um RC interno. Este tempo a mais pode variar de chip para chip, devido ao Vdd, temperatura e processos de variações.

O PWRT timer é habilitado nos fusíveis de proteção, referente ao PWRTE.

Oscilador Start-up Timer (OST)

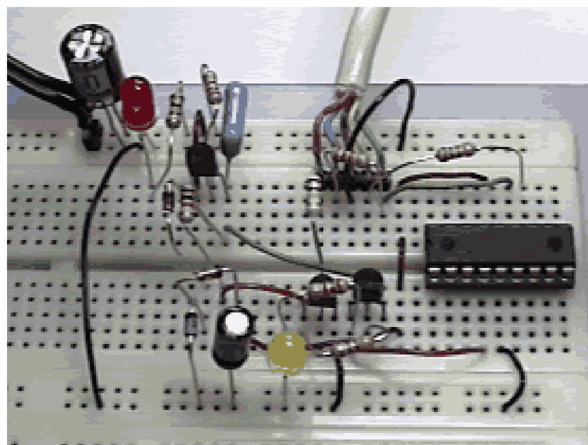
O OST ou tempo de partida do oscilador, gera 1024 pulsos de delay depois do PWRT delay terminar, que é automaticamente habilitado, quando estiver configurado para um dos modos de oscilador XT, LP e HS, e é também válido somente para os resets POR ou o Wake-up gerado pelo SLEEP.

Este procedimento é útil para que o cristal ou ressonador se estabilize, garantindo com que o reset seja bem sucedido, permitindo o uso do circuito mínimo para reset POR.

Por essa aula é só, sabemos que é maçante a parte teórica, mas temos que passar por ela, não tem outra opção. O que vimos é bem pouco, mas se não colocarmos em prática não há asimilação, sendo assim, paramos por aqui, na próxima aula vamos ver como fazer um circuito para gravar o seu PIC. depois do gravador pronto, podemos ir para as próximas aulas, aí começaremos ao software do PIC, mas ainda veremos mais registros, mais hardware do PIC. Mas sempre em doses "homeopáticas".

Construindo um Gravador de PIC

Nesta aula, vamos construir o nosso gravador de pic, o nosso programmer, o circuito que escolhi é uma variação do *JDM programmer* , o circuito original não usa fonte externa, a alimentação é feita pela própria porta serial do PC, funciona muito bem para a linha PIC xxCxxx, e alguns flashs, mas tive problemas com o 16F84A, inclusive demorei bastante para elaboração dessa aula, pois dependendo do PC, a tensão da porta serial não era suficiente para gravar, e em outros funcionava normal, então para nossos alunos não terem tantos problemas, resolvi adaptar uma fonte de tensão externa para suprir os 13 volts necessários para o pic entrar em modo de programação, apesar de aumentar um pouco o circuito, ainda assim é de baixo custo e fácil montagem. Estimativa de custo: aproximadamente US$ 5,00 sem a fonte, sem o Proto Board e sem o PIC. Como experiência eu recomendo montar o gravador no Proto Board, o mesmo que vc vai usar para montar os projetos das próximas aulas, a gente monta num dos cantos, sobrando espaço para os projetos. Vale lembrar, que para gravar o PIC com esse circuito, vc tem que tirar o pic do circuito do projeto, colocar no gravador, gravar, e depois recolocar no projeto.



Gravador montado num ProtoBoard

Uma outra alternativa é montar o circuito numa placa de circuito impresso, ver foto abaixo, isso facilita um pouco mais.

ESQUEMA em ACCEL-EDA (V15.00)

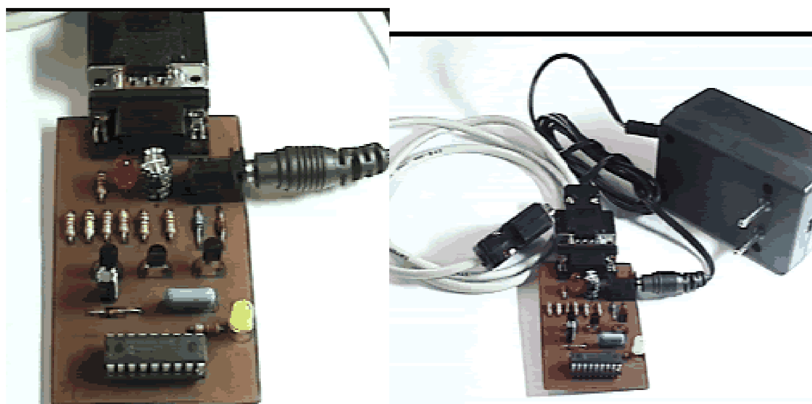
PCB em ACCEL-EDA (V15.00)

Fotos acima: Placa lado do cobre - Componente por cima da placa - Placa Espelhada.

Esse desenho é apenas uma sugestão, vc pode modificar de acordo com suas necessidades...

O Cabo de tranferência para a serial do PC deve ser feito com usando um terminal macho para conectar na placa, e um fêmea para conectar no PC, as ligações são PINO a PINO

Conector Fêmea do Cabo (PLACA)	Conector Fêmea do Cabo (PC)
3	3
4	4
5	5
7	7
8	8



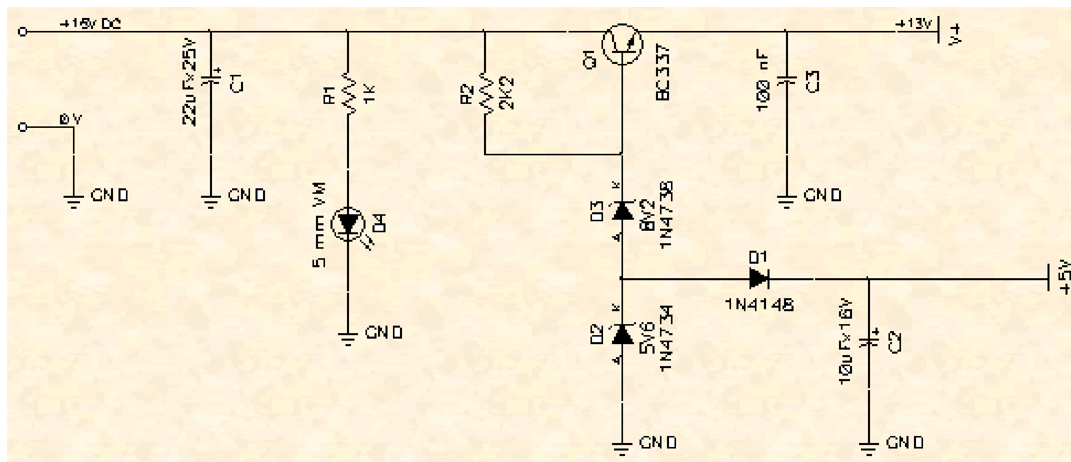
Gravador montado numa placa de circuito impresso.

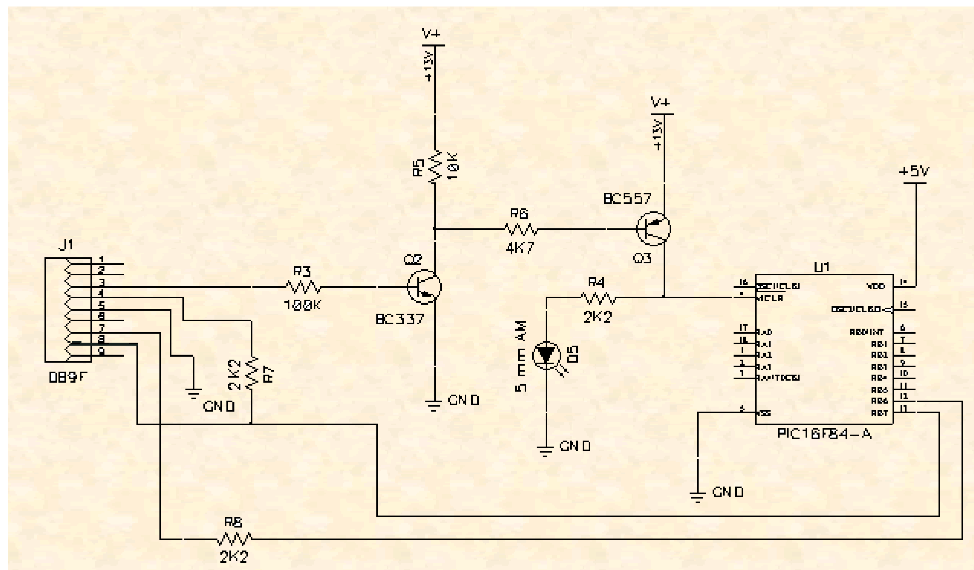
O PIC possui sistema serial de gravação, possuindo um pino para ativação do modo programação, esse é que tem que estar com a tensão alta; um pino de comunicação bidirecional, para gravar e ler os programas; um pino de clock; e logicamente a alimentação normal 5V e GND. Os pinos acima são de multiplas funções, assim no modo normal são usados como /MCLR , RB7, RB6, VDD e VSS.

O nosso gravador usa o software *ICPROG* e grava os seguintes pic´s: 12C508, 12C508A, 12C509, 12C509A, 12CE518, 12CE519, 12C671, 12C672, 12CE673, 12CE674, 16C61, 16C62A, 16C62B, 16C63, 16C63A, 16C64A, 16C65A, 16C65B, 16C66, 16C67, 16C71, 16C72, 16C72A, 16C73A, 16C73B, 16C74A, 16C76, 16C77, 16C84, 16F83, 16F84, 16F84A, 16C505, 16C620, 16C621, 16C622, 16C622A, 16F627*, 16F628*, 16C715, 16F870*, 16F871*, 16F872*, 16F873*, 16F874*, 16F876*, 16F877*,16C923, 16C924

*Para estes pic´s o pino "PGM" deve estar colocado ao GND.

Esquema elétrico do circuito





[Esquema em PDF](#)

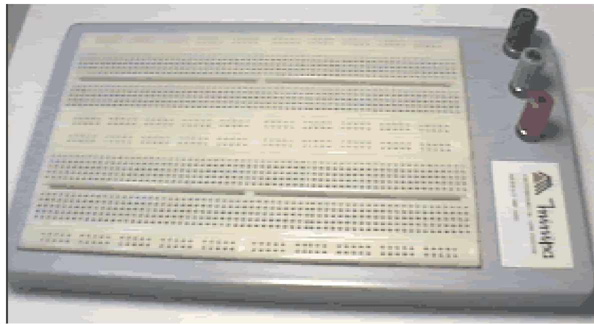
Obs. o conector DB9 no esquema é fêmea

Relação de material

- Fonte de alimentação de 15V DC x 500mA



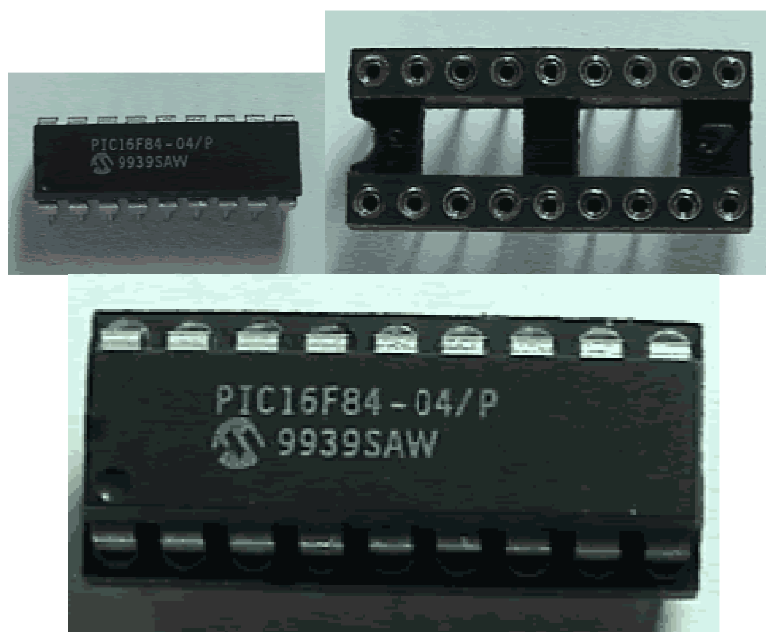
- 1 Proto Board (matriz de contatos), para montagens do gravador e de todos os experimentos.



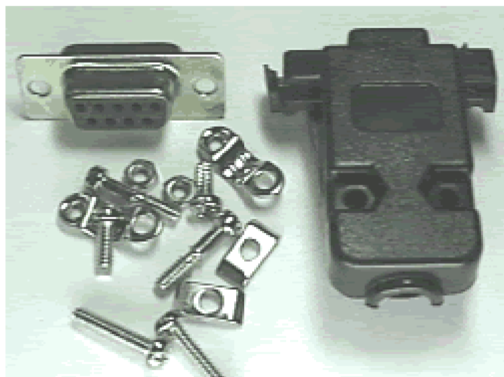
- Fios para ligação no Proto Board (fio rigido de diâmetro aprox. 0,5 mm "par trançado de telefone")



- 1 PIC16F84 ou PIC16F84A (eu aconselho usar um soquete de pino torneado para evitar que se quebre os pinos do PIC, pois você vai ter que manipular o microcontrolador toda vêz de gravá-lo, o soquete vai proteger o pic)



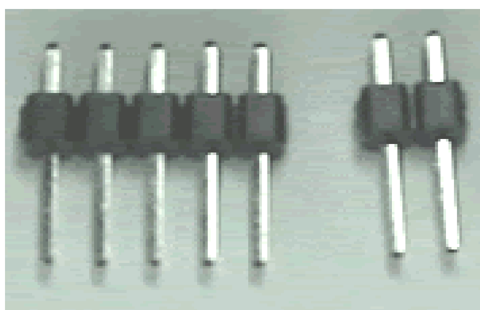
- 1 Conector DB9 fêmea para cabo e
- 1 Capa plástica para conector DB9 cabo



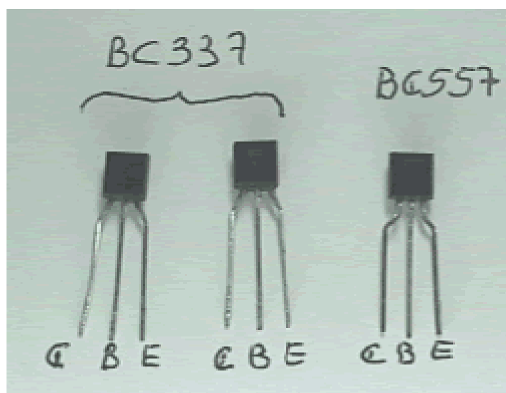
- 1,5m Cabo manga de pelo menos 5 vias (cabo para fazer a conexão do gravador ao PC)



- 2 Barra de pinos (5 vias p/ cabo serial e 2 vias para a fonte DC)

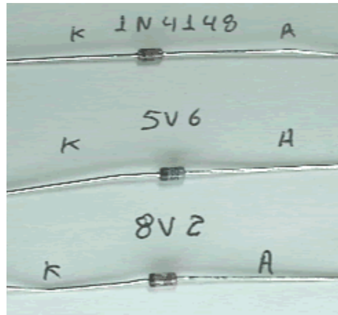


- 2 Transistor BC337 ou equivalente NPN
- 1 Transistor BC557 ou equivalente PNP

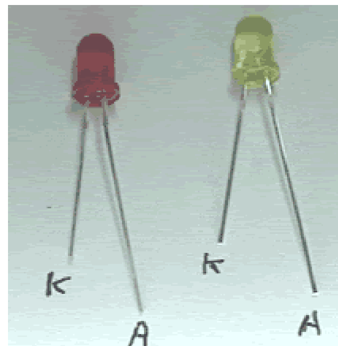


- 1 Diodo 1N4148
- 1 Diodo zener de 5V6

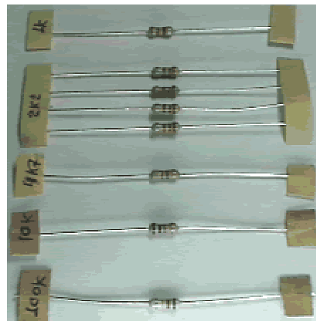
- 1 Diodo zener de 8V2



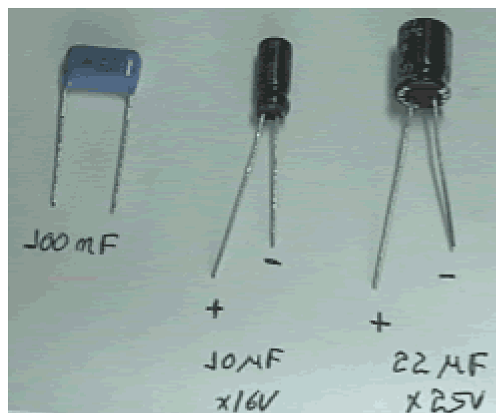
- 1 LED vermelho (indicação de fonte ligada)
- 1 LED amarelo (indicação de gravando)



- 1 Resistor de 1K
- 4 Resistor de 2K2
- 1 Resistor de 4K7
- 1 Resistor de 10K
- 1 Resistor de 100K

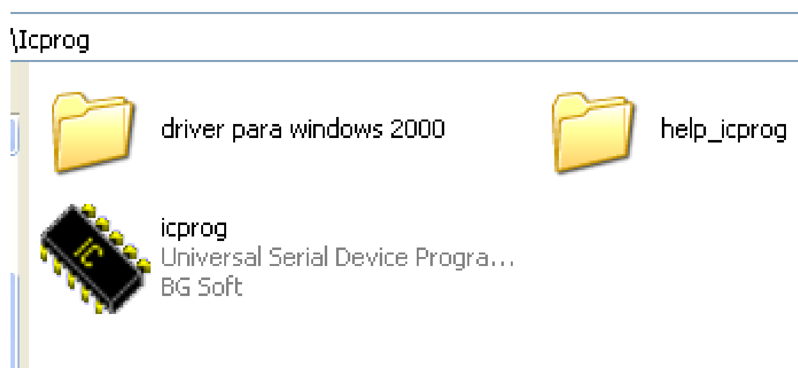


- 1 Capacitor de 100nF (nano Farady)
- 1 Capacitor de 22uF x 25V (micro Farady)
- 1 Capacitor de 10uF x 16V(micro Farady)



Instalando o software ICPROG no seu Microcomputador

Crie uma pasta por exemplo, icprog na unidade C: e copie os arquivos descompactados. depois disso clique no ícone do ICprog. Eu aconselho você criar um atalho para ele na área de trabalho do seu micro, pois vai ser bastante usado.



PASSO 2

Vai aparecer a tela abaixo, se for a primeira vez que está usando o icprog. Clique em OK



PASSO 3

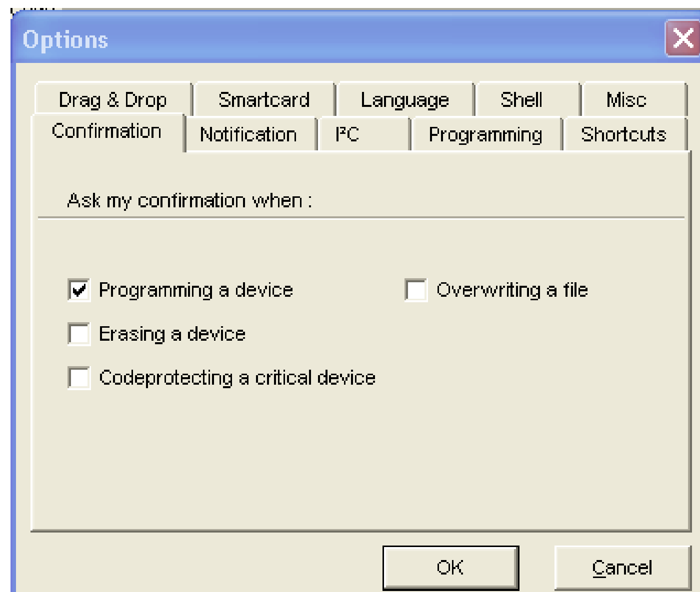
Você vai configurar o seu software para a porta serial do seu microcomputador que vai ser usada para o gravador, com1 ou com2 conforme o seu micro. Se você estiver usando o windows XP ou 2000, selecione a Interface Windows API. O restante deixe como está, O I/O delay (10) nunca me deu problemas, mas ele é responsável pela velocidade de transmissão do arquivo hexadecimal para o PIC. Clique em OK

PASSO 4

Deve aparecer a tela principal do software, vamos alterar sua linguagem para o português. Vá em Settings > Options.

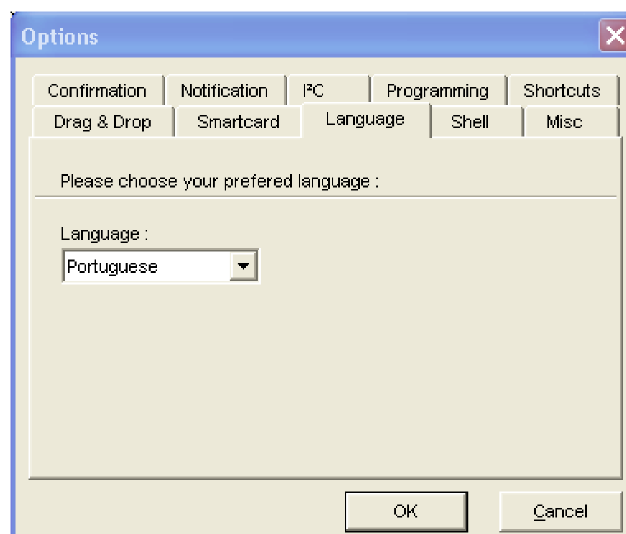
PASSO 5

Clique em Language



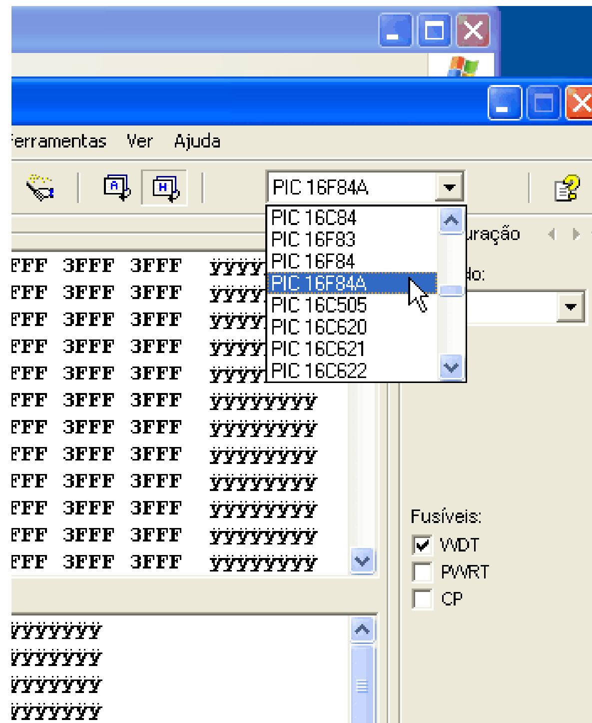
PASSO 6

Selecione Portuguese. E clique em OK



PASSO 7

Agora vamos selecionar o PIC 16F84A



Pronto! o IcProg está instalado

---

*Se você está usando Windows NT, 2000, ME ou XP, **clique aqui** para configurar corretamente o icprog.*

PASSO 8

Vamos gravar um arquivo chamato teste.hex clique aqui para o download ele está zipado, descompacte-o numa pasta de teste.

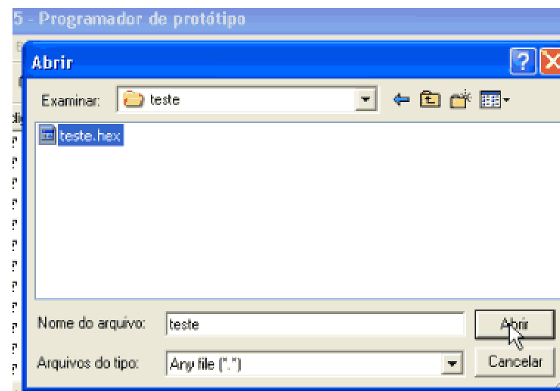
Vamos gravar o software no PIC . Antes de tudo conecte a fonte de alimentação no gravador, e o cabo serial no seu microcomputador. O led vermelho tem que estar aceso.

Ligou? Então primeiro vá em Arquivo > Abrir



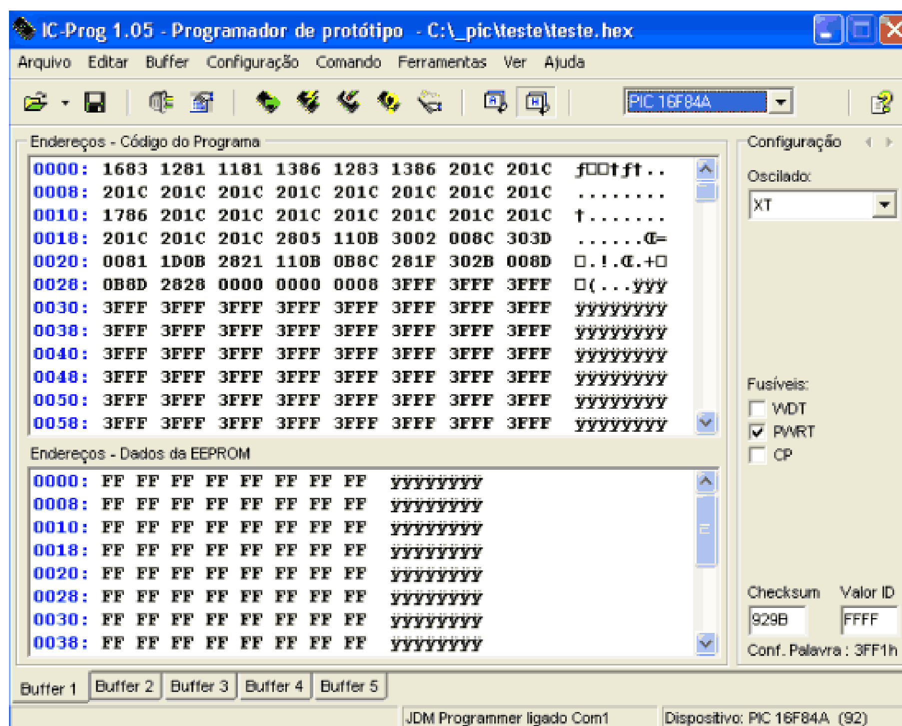
PASSO 8.1

Selecione o arquivo TESTE.HEX que você baixou e clique em abrir



PASSO 8.2

Vai aparecer a tela abaixo, o que está vendo é o código hexa do nosso software do PIC.



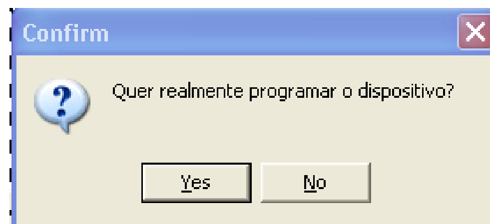
PASSO 8.3

Clique no ícone do CI com um raio, isso dispara o processo para gravar o PIC.



PASSO 8.4

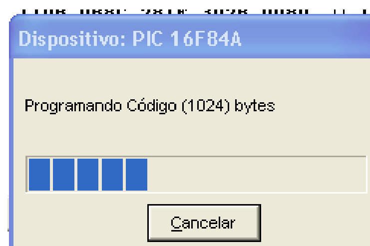
Clique em Yes



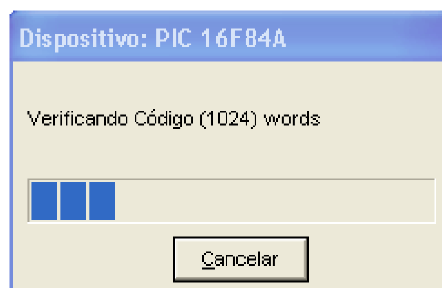
O Led amarelo do gravador vai piscar no começo e depois vai acender...

PASSO 8.5

Aguarde o processo de gravação da programação



Aguarde a verificação dos dados gravados



Se deu tudo certo vai aparecer a seguinte janela: clique em OK.



Se deu alguma coisa errada vai aparecer a seguinte tela:



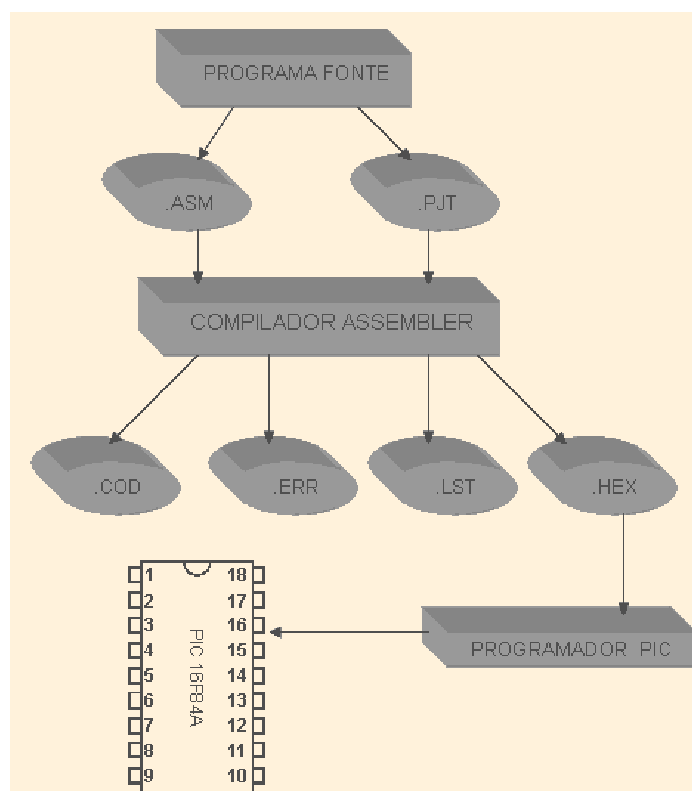
Isso normalmente ocorre se você esqueceu de ligar a fonte, ou o cabo serial. caso não seja este o seu caso tente de novo, confira todo o circuito novamente.

Software para o PIC

Todo sistema microprocessado ou microcontrolado, necessita de um programa (software) para comandá-lo. O microcontrolador irá obedecer fielmente todas as ordens que forem atribuídas.

Um programa é constituído por um conjunto de instruções em seqüência, onde cada uma identificará precisamente a função básica que o PIC irá executar. Cada instrução é representada por um código de operação (OPCODE - do inglês, Operation Code) de 14 bits, tamanho exato de cada locação de memória de programa, no caso do PIC 16F84.

O programa será escrito através de instruções mnemônicas (lembrando que o PIC 16F84 possui 35), podendo ser utilizado um editor de texto comum, ou como no nosso caso, um ambiente de desenvolvimento como o Mplab. Logo após a edição do programa fonte, será feita a compilação (transformar a linguagem de texto em linguagem apropriada para que o PIC possa entender) e finalmente gravar o PIC. A figura abaixo mostra o fluxograma das operações necessárias até a gravação do PIC (utilizando o MPLAB).



PROGRAMAÇÃO ASSEMBLER

A linguagem de programação Assembler do PIC 16F84 é composto por 35 instruções. As instruções são expressas na forma de mnemônicos. O mnemônico é composto por verbos e argumentos. Os verbos definem as ações que são completadas e especificadas pelos argumentos.

Verbos ou termos (mnemônicos)	(inglês)	Ação do verbo ou tradução do termo
ADD	*add*	Adicionar

AND	*and*	Fazer um "E" lógico
B	*bit*	Um bit de um file(file=posição de RAM)
C	*clear*	Limpar, colocar em zero
CALL	*call*	Chamar
CLR	*clear*	Limpar
COM	*complement*	Complementar (exemplo: o complemento de 1 é zero)
d	*destination*	Refere em qual file de destino vais ser armazenado o resultado da operação: se d=0 é aemazenado em *w(registro de trabalho),* se d=1 é armazenado no *f (file)*
DEC	*decrement*	Decrementar um
F	*file*	Registro, é uma posição da RAM
GOTO	*go to*	Vai para...
INC	*increment*	Incrementar um
IOR	*inclusive or*	Fazer um "ou inclusivo" lógico
k	*constant*	É uma contante, pode ser um literal ou um label
L	*literal*	É uma constante
MOV	*move*	Mover
NOP	*no operation*	Não faça nenhuma operação
RETFIE	*return from interrupt*	Retornar de uma interrupção
RETLW	*return with literal in w*	Retornar com um valor literal em w
RETURN	*return*	Retornar de uma subrotina
RL	*rotate right*	Rodar para direita
RR	*rotate left*	Rodar para esquerda
S	*set ou skip*	Set= colocar em 1 / Skip= ação de pular
SLEEP	*sleep*	Entrar em modo standby, ou modo de espera, (economia de energia)
SUB	*subtract*	Subtrair
SWAP	*swap*	trocar
T	*test*	Testar
W	*work*	É o file de trabalho,(registro de trabalho)
WDT	*watch dog timer*	É o registro de RAM onde está o temporizador do periférico "watch dog"
XOR	*exclusive or*	fazer um "ou exclusivo" lógico
Z	*zero*	Zero

Conjunto de instruções do PIC16F84

As instruções são divididas em três grupos:

• instruções orientadas a byte (registradores);
• instruções orientadas a bit;
• instruções com constantes (literais) e de controle.

Instrução	Argu- mentos	Descrição	Ciclos	OPCODE 14-Bit														Bits afetados no STATUS
				Bit														
				13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Operações de bytes com registros																		
ADDWF	f,d	Adicionar *w* e *f*	1	0	0	0	1	1	1	d	f	f	f	f	f	f	f	C,DC,Z
ANDWF	f,d	Faz um AND lógico entre *w* e *f*	1	0	0	0	1	0	1	d	f	f	f	f	f	f	f	Z
CLRF	f	Faz todos os bits de *f* = zero	1	0	0	0	0	0	1	1	f	f	f	f	f	f	f	Z
CLRW		Faz todos os bits de *w* = zero	1	0	0	0	0	0	1	d	x	x	x	x	x	x	x	Z
COMF	f,d	Complementa *f* (inverte os bits de 0>1 ou 1>0)	1	0	0	1	0	0	1	d	f	f	f	f	f	f	f	Z
DECF	f,d	Decrementa uma unidade em *f*	1	0	0	0	0	1	1	d	f	f	f	f	f	f	f	Z
DECFSZ	f,d	Decrementa uma unidade em *f* e pula a próxima instrução se *f* = zero	1 ou 2	0	0	1	0	1	1	d	f	f	f	f	f	f	f	-
INCF	f,d	Incrementa uma unidade em *f*	1	0	0	1	0	1	0	d	f	f	f	f	f	f	f	Z
INCFSZ	f,d	Incrementa uma unidade em *f* e pula a próxima instrução se *f* = zero	1 ou 2	0	0	1	1	1	1	d	f	f	f	f	f	f	f	-
IORWF	f,d	Faz um OU lógico entre *w* e *f*	1	0	0	0	1	0	0	d	f	f	f	f	f	f	f	Z
MOVF	f,d	Move *f*	1	0	0	1	0	0	0	d	f	f	f	f	f	f	f	Z
MOVWF	f	Move *w* para *f*	1	0	0	0	0	0	0	1	f	f	f	f	f	f	f	-
NOP		Nenhuma operação	1	0	0	0	0	0	0	0	x	x	0	0	0	0	0	-
RLF	f,d	Roda o byte à esquerda passando pelo Carry	1	0	0	1	1	0	1	d	f	f	f	f	f	f	f	C
RRF	f,d	Roda o byte à direita passando pelo Carry	1	0	0	1	1	0	0	d	f	f	f	f	f	f	f	C
SUBWF	f,d	Subtrai *w* de *f*	1	0	0	0	0	1	0	d	f	f	f	f	f	f	f	C, DC, Z
SWAP	f,d	Troca nibles em *f*	1	0	0	1	1	1	0	d	f	f	f	f	f	f	f	-
XORWF	f,d	Faz um XOR lógico entre *w* e *f*	1	0	0	0	1	1	0	d	f	f	f	f	f	f	f	Z
Operações de bits com registro																		
BCF	f,b	Zera o bit b em f	1	0	1	0	0	b	b	b	f	f	f	f	f	f	f	-
BSF	f,b	Seta,(1) o bit b em f	1	0	1	0	1	b	b	b	f	f	f	f	f	f	f	-
BTFSC	f,b	Testa o bit b em f e pula a próxima instrução se o bit for zero	1 ou 2	0	1	1	0	b	b	b	f	f	f	f	f	f	f	-
BTFSS	f,b	Testa o bit b em f e pula a próxima instrução se o bit for 1	1 ou 2	0	1	1	1	b	b	b	f	f	f	f	f	f	f	-
Operações com constantes e de controle																		
ADDLW	k	Adiciona *w* e *k*	1	1	1	1	1	1	x	k	k	k	k	k	k	k	k	C, DC, Z
ANDLW	k	AND lógico entre *w* e *k*	1	1	1	1	0	0	1	k	k	k	k	k	k	k	k	Z
CALL	k	Chama a sub rotina *k*	2	1	0	0	k	k	k	k	k	k	k	k	k	k	k	-
CLRWDT		Zera o timer do watch dog	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	TO, PD
GOTO	k	Vai para o label *k*	2	1	0	1	k	k	k	k	k	k	k	k	k	k	k	-
IORLW	k	Faz um OR lógico entre *w* e *k*	1	1	1	1	0	0	0	k	k	k	k	k	k	k	k	Z
MOVLW	k	Move *l* para *w* (w = k)	1	1	1	0	0	x	x	k	k	k	k	k	k	k	k	-
RETFIE		Retorna da interrupção	2	0	0	0	0	0	0	0	0	0	0	1	0	0	1	-
RETLW	k	Retorna com *w* = *k*	2	1	1	0	1	x	x	k	k	k	k	k	k	k	k	-
RETURN		Retorna de subrotina	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-

SLEEP		Entra em modo standby	1	0	0	0	0	0	0	0	1	1	0	0	0	1	1	TO, PD
SUBLW	k	Subtrai *w* de *l*	1	1	1	1	1	0	x	k	k	k	k	k	k	k	k	C, DC, Z
XORLW	k	Faz um XOR lógico	1	1	1	1	0	1	0	k	k	k	k	k	k	k	k	Z

A tabela acima mostra o conjunto de instruções do PIC 16F84, com os mnemônicos das instruções, operandos, descrição de cada instrução, ciclos gastos, código binário de 14 bits das instruções e bits de flags envolvidos. Para entender melhor a tabela, abaixo está o esclarecimento de algumas letras que compõe o argumento:

- f - File, registro que está sendo acessado, é uma posição da memória RAM de dados, como tem apenas 7 bits esse número vai de zero a 127, mas com o recurso do MPlab, usando a diretiva EQU, *equate* , podemos escrever os nomes dod registros que a Microchip sugere, como por exemplo: STATUS, OPTION, TRISB, com o equate o compilador substitui o nome pelo número na hora de gerar o código hexadecimal.
- d - Destino, indica para onde deve ir o resultado da operação. Devido ao conjunto reduzido de instrução e uma instrução já conter o opcode, operando e o destino, só existe duas possibilidades de destino do resultado:
  - d = 0, o resultado será armazenado no registro W
  - d = 1, o resultado será armazenado no registro f
- b - Indica qual bit do registro esta sendo acessado, podendo usar o mnemônico do bit, (da mesma forma do f com o equatre), ou através do numero do bit (0 - 7), note que há apenas três bits disponiveis nas isnstruções orientadas a bits.
- K - Pode ser uma constante númerica, , ou uma indicação de posição de memória de programa:
  - Constante numérica de 8 bits (0 - 255), valor literal para as operações matemáticas e lógicas;
  - Indicação de posição, apenas nas instruções CALL e GOTO, possui 11 bits (0 - 2047), pode ser um número imediato, ou um label, nesse caso toda vêz que o compilador encontra um label no programa ele troca por um número na sequência crescente. Por exemplo: Numa instrução GOTO repete, se o repete é o primeiro label o compilador troca por GOTO 000
- x tanto faz, pode ser zero ou 1, o compilador da Microchip gera o zero, e recomenda isso.

Constantes para o Compilador

Ao representar números decimais, hexadecimais e binários na programação, devemos escrever da seguinte forma:

- Constante decimal - D'valor' ou d'valor'. exemplo: 55 decimal = d'55' ou ainda D'55'.
- Constante Hexadecimal - 0Xvalor ou valorH. Exemplo: 3B hexa = 0x3B ou 3Bh ou 3BH, *Importante*: No caso da constante começar com letra devemos colocar colocar um zero antes. Exemplo: E3h fica 0E3h).
- Constante binária - B'valor' ou b'valor'. Exemplo: 10101010 binário = b'10101010'

Descrição das Instruções

Como foi visto na tabela, as instruções estão divididas em 3 partes: Instruções orientadas a byte, instruções orientadas a bits e instruções de operações com controles e constantes.

Instruções Orientadas a Bytes

Cada tipo de instruções descritas tem um formato diferente, quanto à formação da palavra (instrução). É esse codigo binário, que vai "dizer" à CPU do PIC qual a operação e o argumento para executar. Notamos então, que num único OPCODE, código de operação, temos todas as

informações necessárias para o PIC executar uma instrução completa, (instrução - argumento - destino)

OPCODE Instrução de 14 bits (orientadas a bytes)														
13	12	11	10	9	8	7	6	5	4	3	2	1	0	Bit
Código da operação						destino	File (posição do byte na RAM)							função
0	0	*x*	*x*	*x*	*x*	d	f	f	f	f	f	f	f	valor

ADDWF f,d

Faz a soma entre W e f (d = W + f). O valor de W será somado ao conteúdo do registro f. Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W

Exemplo: Se W = 20 (decimal), e f(20h) = 15 (decimal)

A instrução ADDWF 20h,0 - resultará W = 35

A instrução ADDWF 20h - resultará f(20h) = 35

ANDWF f,d

Faz a operação lógica AND entre W e f (d = W AND f). A operação lógica será feita entre o conteúdo de W e o conteúdo do registro f. Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W.

Exemplo : Se W = b'01011001', e f(0Ch) = b'00110111'

A instrução ANDWF 0Ch,0 - resultará em W = b'00010001'

A instrução ANDWF 0Ch - resultará em f(0Ch) = b'00010001'

W = 01011001
AND f(0Ch) = 00110111
Resultado = 00010001

CLRF f

Zera o conteúdo do registro f indicado.

Exemplo: se TRISB = b'00101010'
-
A instrução CLRF TRISB - resulta em TRISB = b'00000000'

Fazendo o flag Z = 1

CLRW

Zera o conteúdo de W

Exemplo: se W = 0x3F
-
A instrução CLRW - resulta em W = 0x00

Fazendo o flag Z = 1

COMF f,d

Complementa f. Os bits do registro f indicado serão complementados, ou seja, invertidos. Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W

Exemplo: Se f(12h) = 2Bh = b'00101011'

A instrução COMF 12h - resultará em f(12h) = D4h = b'11010100'

A instrução COMF 12h,0 - resultará em W = D4h , sem alterar o conteúdo o conteúdo do f(12h).

DECF f,d

Decrementa f. Decrementa o conteúdo do registro f em -1 a cada instrução executada (d = f -1). Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W.

Exemplo: Se f(1Fh) = 20 (decimal)
-
A instrução DECF 1Fh,1 - resultará em f(1Fh) = 19

A instrução DECF 1Fh,0(W) - resultará em W = 19, sem alterar o registro f(1Fh).
Em qualquer instrução f,d onde for indicado para armazenar em W, pode ser colocado d = 0, ou d = W.

DECFSZ f,d

Decrementa f e pula se f = 0. Decrementa o conteúdo do registro f em -1 (d = f - 1) a cada instrução executada, e ainda testa o registro se chegou a zero. Se o registro chegar a zero, pula uma linha de instrução. Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W.

Exemplo: Se o registro f(0Dh) = 02h, e as seguintes linhas de instrução for executada:

```
.
.
Decrementa
DECFSZ 0Dh ; Decrementa o registro f(0Dh)
GOTO decrementa ;Vai para decrementa enquanto 0Dh não for igual a zero
END ;Se 0Dh = 0 termina o programa
```

Obs: Esta instrução levará dois ciclos de máquina se o registro f resultar 0, pois a próxima instrução será executada como um NOP (GOTO decrementa).

INCF f,d

Incrementa f. Esta instrução incrementa o conteúdo do registro f em +1 (d = f + 1). Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W.

Exemplo: Se o registro f(0Ch) = 20

A instrução INCF 0Ch - resultará em f(0Ch) = 21

A instrução INCF 0Ch,0 - resultará em W = 21, sem alterar o registro 0Ch

---

INCFSZ f,d

Incrementa f, e pula se f = 0. Esta instrução irá incrementar o conteúdo do registro f em + 1 (d = f + 1) e o testará , pulando uma linha de instrução se o conteúdo do registro for igual a zero. Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W.

Exemplo: Se o registro f(0Ch) = 254(decimal), e as instruções abaixo forem executadas:

```
.
.
Incrementa:
INCF 0Ch,1 ; Incrementa o conteúdo do registro 0Ch em +1
GOTO incrementa ; vai para incrementa enquanto 0Ch # 0
END ;Quando o conteúdo do registro 0Ch chegar a zero termina o programa.
```

Obs: Esta instrução levará dois ciclos de máquina se o registro f resultar 0, pois a próxima instrução será executada como um NOP (GOTO decrementa).

IORWF f,d

Função OR entre W e f. Esta instrução executará a função lógica OR entre W e o registro f (d = W OR f). Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W.

Exemplo: Se o conteúdo de W = 3Ah (b'00111010'), e o conteúdo do registro f(0Eh) = A3h (b'10100011').

A instrução IORWF 0Eh - resultará em f(0Eh) = BBh - b'10111011'

A instrução IORWF 0Eh,0 - resultará em W = BBh

W = 00111010
OR f(0Eh) = 10100011
Resultado = 10111011

MOVF f,d

Move para f. Esta instrução transfere o conteúdo do registro f para W ou para ele mesmo (d = f). Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W.

Exemplo: Se o conteúdo do registro f(20h) = 55h

A instrução MOVF 20h,W - resultará em W = 55h

A instrução MOVF 20h - obviamente não irá alterar o registro f(20h).

Se f(20h) = 0;

A instrução MOVF 20h - irá afetar o flag de zero Z, fazendo igual a 1. Essa é uma forma de conseguir setar o bit de flag Z sem alterar o valor contido em W.

MOVWF f

Move W para f. Transfere o conteúdo de W para o registro f (faz f = W).

- Exemplo: Se o conteúdo de W = 5Ah

A instrução MOVWF f(25h) - Resultará em conteúdo do registro f(25h) = 5Ah

---

NOP

Nenhuma operação. Esta instrução só serve para gastar tempo equivalente a um ciclo de máquina.

RLF f,d

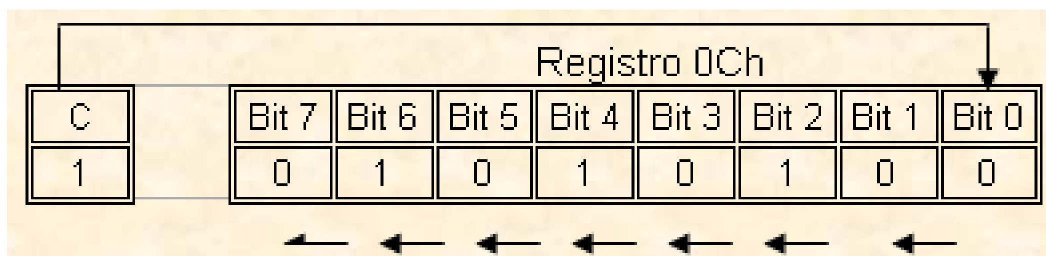
Desloca uma vez os bits do registro f à esquerda passando através do flag de carry C. Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W.

Exemplo: Se o conteúdo de f(0Ch) = b'10101010'.



Registro 0Ch

A instrução RLF 0Ch - resultará no registro 0Ch = b'01010100'



Esta instrução afeta o flag de Carry.

RRF f,d

Desloca uma vez os bits do registro a direita passando pelo flag de carry C. Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W.

SUBWF f,d

Subtrai W de f. Faz operação aritmética de subtração do conteúdo de W do conteúdo do registro f (d = f - W). Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W.

Exemplo: Se o conteúdo de W = 25 e o conteúdo do registro f(20h) = 50.

A instrução SUBWF 20h - resultará em f(20h) = 25 (positivo).

A Instrução SUBWF 20h,0 - resultará em W = 25 (positivo).

O flag de Carry C = 1 (positivo).

Se o conteúdo de W = 50 e o conteúdo do registro f(20h) = 25

A instrução SUBWF 20h - resultará em f(20h) = - 25 = E7h (negativo)

O flag de carry C = 0 (negativo)

SWAPF f,d

Troca os nibbles em f. Esta instrução troca os 4 bits mais significativos pelos 4 menos significativos no registro f . Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W.

- Exemplo: Se o conteúdo do registro f(20h) = 5Ah.

Registro 20h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	1	0	1	1	0	1	0

A instrução SWAPF 20h resultará em f(20h) = A5h

Registro 20h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	1	0	0	1	0	1

Nibble - conjunto de 4 bits

XORWF f,d

OU exclusivo entre W e f. Realiza a operação lógica OU EXCLUSIVO entre o conteúdo de W e o conteúdo do registro f (d = f W). Se d = 1, o resultado será armazenado no registro. Se d = 0 será armazenado em W.

Exemplo: Se W = 55h e o registro f(20h) = 55h

A instrução XORWF 20h - resultará em f(20h) = 00h

W = 01010101
F(20h) = 01010101
Result. =00000000

A instrução XORWF 20h,0 - resultará em W = 00h

O resultado acima afetará o flag Z = 1.

Com esta instrução é possível comparar se dois números são iguais, testando o flag de zero Z.

Instruções Orientadas a Bits

Na instruções orientadas a bits o formato como na figura abaixo. O formato da palavra passa a utilizar três bits para seleção dos bits contidos nos registros. A grande vantagem de trabalhar com bits independentes, é que podemos alterar somente o bit desejado do registro, sem a necessidade de formar um número de 8 bits para saber como devemos coloca-los no registro

OPCODE Instrução de 14 bits (orientadas a bite)														
13	12	11	10	9	8	7	6	5	4	3	2	1	0	Bit
Código da Operação				Número do bit			File (posição do byte na RAM)							função
0	1	*x*	*x*	*b*	*b*	b	f	f	f	f	f	f	f	valor

BCF f,b

zera o bit de f. Esta instrução faz o bit (0 - 7) indicado por 'b', do conteúdo do registro f ir para o nível lógico 0.

Exemplo: Se o conteúdo do registro TRISB = 10101111

A instrução BCF TRISB, 7 - resultará em TRISB = 00101111

BSF f,b

Seta o bit de f. A instrução faz o bit indicado por 'b', do registro f ir para nível lógico 1.

Exemplo: S o conteúdo do PORTB = 00000000.

A instrução BSF PORTB, 5 - resultará em PORTB = 00100000

BTFSC f,b

Testa o bit de f e pula se b = 0. Esta instrução testa o bit indicado por 'b', do registro f, se este bit for zero pula uma instrução, Caso contrário executa a seguinte.

Exemplo:

```
Teste
BTFSC PORTB,0 ; Testa a o pino RB0
GOTO TESTE ; RB0 = 1, testa novamente
GOTO LIGA ; RB0 = 0 vai para rotina LIGA
```

BTFSS f,b

testa o bit de f e pula se b = 1. Ao contrário da instrução anterior, esta testa o bit indicado por 'b', do registro f, e se este bit for 1 pula uma instrução, se 0 executa a seguinte.

Exemplo:

Teste2:
BTFSS PORTB,3 ;Testa o pino RB3
GOTO TESTE2 ; se RB3 = 0, testa novamente RB3
GOTO DESLIGA ; se RB3 = 1 vá para rotina DESLIGA

Instruções Orientadas a Constantes e de Controle

Na instruções orientadas a constantes como na figura abaixo. O formato da palavra passa a utilizar oito bits para as constantes numéricas que são os dados dos registros, ou onze bits se fizer referência a endereço de memória de programa. Se for apenas de controle, sem envolver dados ou endereço de programa, é só o código da operação.

OPCODE Instrução de 14 bits (orientadas a constante númerica)														
13	12	11	10	9	8	7	6	5	4	3	2	1	0	Bit
Código da Operação						Constante númerica de 8 bites (0 - 255)								função
1	1	*x*	*x*	*x*	*x*	k	k	k	k	k	k	k	k	valor

OPCODE Instrução de 14 bits (orientadas a constante que indica posição de memória de programa)														
13	12	11	10	9	8	7	6	5	4	3	2	1	0	Bit
Código da Operação			Constante númerica de 11 bites (0 - 2047)											função
1	0	*x*	*k*	*k*	*k*	k	k	k	k	k	k	k	k	valor

OPCODE Instrução de 14 bits (orientadas a controle)														
13	12	11	10	9	8	7	6	5	4	3	2	1	0	Bit
Código da Operação														função
0	0	0	0	0	0	0	x	x	0	x	x	x	x	valor

ADDLW k

Soma W a constante k. Esta instrução somará o conteúdo de W a uma constante (literal) imediata k.

Exemplo: Se o conteúdo de W = 05, e se deseja somar um valor imediato 20.

A instrução ADDLW 20 - resultará em W = 25 (W = 05 + 20)

Lembrando que 0 =< k < = 255

ANDLW k

AND entre W e constante k. A instrução executará uma função lógica AND entre o conteúdo de W e a constante imediata k.

Exemplo: Se o conteúdo de W = 01011011 e se de seja fazer uma lógica com um valor k=00000001.

A instrução ANDLW b'00000001' - resultará em W = 00000001

W = 01011011
AND k = 00000001
Resultado = 00000001

Esta é uma forma de se isolar um bit (mascaramento), no caso o bit 0.

CALL k

Chama sub-rotina. A instrução CALL chama uma sub-rotina endereçada por k, com isso o contador de programa PC é atualizado com o valor de K, salvando no STACK o endereço de retorno da sub-rotina.

Exemplo: no endereço 40h da memória de programa chama uma sub-rotina localizada no endereço 100h.

```
.
.
40h: CALL 100h ; chama a sub-rotina, e o endereço 40h + 1 é salvo no STACK, e o PC são
carregado com o valor 100h
41h: xxxx xxx ; Uma instrução qualquer
.
.
.
100h: XXXXXX ; Uma instrução qualquer
101h: RETURN ; Retorna da sub-rotina, carregando o PC com o valor do STACK - 41h
```

Importante: Todas as vezes que ocorrer um CALL, o endereço atual é salvo no STACK para ser recuperado mais tarde. O PIC aceita até 8 chamadas consecutivas, ocorrendo mais um CALL sem o retorno de nenhum, ocorrerá um OVERFLOW (estouro da capacidade) no STACK, Prejudicando o programa.

CLRWDT

Zera o Timer do WatchDog. Esta instrução irá zerar o conteúdo do temporizador Watch Dog, evitando um reset se este estiver habilitado.

Exemplo: se WDT está com um tempo de 10 ms.

A instrução CLRWDT - resultará em WDT = 0 ms

Dois bits serão afetados no STATUS: e que serão setados.

GOTO k

Desvia para o Label k. Esta instrução faz com que o programa desvie para um novo endereço indicado por k, com isso o valor do PC é atualizado com o valor de k, é o endereço anterior não é salvo no STACK.

Exemplo: suponha que no endereço 30h da memória de programa ocorra um GOTO 55h.
.
.
30h: GOTO 55h ; desvia para o endereço 55h
.
.
.
.
55h: xxxxx xxxxx ; o programa passa a ser executado deste ponto

---

IORLW k

OR entre W e k. A instrução realizará a função lógica entre o conteúdo de W e um valor imediato k.

Exemplo: Se W = b'01001100'.

A instrução IORLW b'11110000' - resultará em W = b'11111100'

W = 01001100
OR k = 11110000
Resulta W =11111100

MOVLW k

faz W = k. Esta instrução faz o registro W assumir o valor imediato indicado por k. Esta instrução é muito utilizada, principalmente quando queremos alterar o conteúdo de algum registro, ou seja, toda vez que se queira inserir um valor imediato em um registro que não seja o W, esta instrução será necessária.

Exemplo: Vamos fazer o conteúdo de um registro f(20h) = 55h.

MOVLW 0x55 ; Aqui faz W = 55h
MOVF 20h ; Transfere o valor 55h para o registro f(20h): f(20h) = 55h

---

RETFIE

Retorna da interrupção. Toda vez que o PIC atender uma rotina de interrupção, ao final desta deverá ter está instrução.

Exemplo:
004h: MOVWF W_TEMP ;endereço 004h, vetor de interrupção
005h: xxxxxx xxxxx ; instruções qualquer

.
.
00Fh: RETFIE ; Retorna da interrupção, atualizando o PC com o valor armazenado no STACK PC = 102h
.
trecho em execução
0100h: NOP ; Instrução qualquer
0101h: MOVLW xx ** ; ** ocorreu um pedido de interrupção. Salva o
0102h: xxxxx xxx ; próximo endereço no STACK e atualiza o PC com o endereço do vetor, PC = 004h

---

RETLW k

- retorna com o valor de k. a execução desta instrução faz o conteúdo de W, retornar de uma sub-rotina com o valor imediato indicado por k. Esta instrução é muito usada para teste por exemplo de códigos, é testado o valor do código, se é realmente o desejado , o W retorna com o valor do código.

Exemplo:

0100h: CALL decodifica ; Chama rotina decodifica
.
.
.
decodifica:
movwf código ;salva W em registro denominado código

movlw b'11101101' ;testa se é 1
xorwf codigo,w ; compara se os valores são iguais com a instrução XORWF
btfsc status,z ;Testa o Flag de Z, se 0 salta uma instrução
retlw b'00001110' ; se chegar aqui, Z=1, retorna com o valor decodificado indicado por k = b'00001110'

RETURN

retorna da sub-rotina. Esta instrução está sempre no fim de uma sub-rotina chamada pela instrução CALL, exceto quando utiliza a instrução citada anteriormente.

Exemplo:

0100h: CALL 155h ;chama a sub-rotina localizada no endereço 155h salvando o endereço seguinte do PC no STACK (101h)
.
0155h: BTFSS STATUS,Z ;uma instrução qualquer
0156h: GOTO ZERA ;vai realizar uma operação qualquer
0157h: RETURN ;retorna para endereço salvo no STACK (101)

SLEEP

Entra no modo SLEEP. A instrução faz com que o PIC entre em baixo consumo, o oscilador pára, Watch Dog e Prescaler são zerados.

Exemplo:

0200h: SLEEP ; Entra em modo SLEEP

Os flags = 1 e = 0

SUBLW k

subtrai k de W. Esta instrução subtrai o conteúdo do registro W pelo valor imediato indicado por k (W = k - W).

Exemplo: Se o conteúdo de W = 25

A instrução SUBLW 26 - resultará em W = 1, C = 1 (STATUS) indicando um número positivo

A instrução SUBLW 24 - resultará em W = FFh, C = 0 indicando um número negativo, o que corresponde a -1.

Para se chegar ao valor negativo como no exemplo, basta aplicar complemento de dois. W = FF, em binário 11111111.

11111111 complementa 00000000
soma + 1
resultado 00000001 -1

XORLW k

exclusive OR entre W e k. Faz a função lógica OU exclusivo entre o conteúdo de W, e um valor imediato indicado por k (W = W k).

Exemplo: se o conteúdo de W = b'01100110'.

A instrução XORLW b'10010010' - resultará em W = b'11110100'

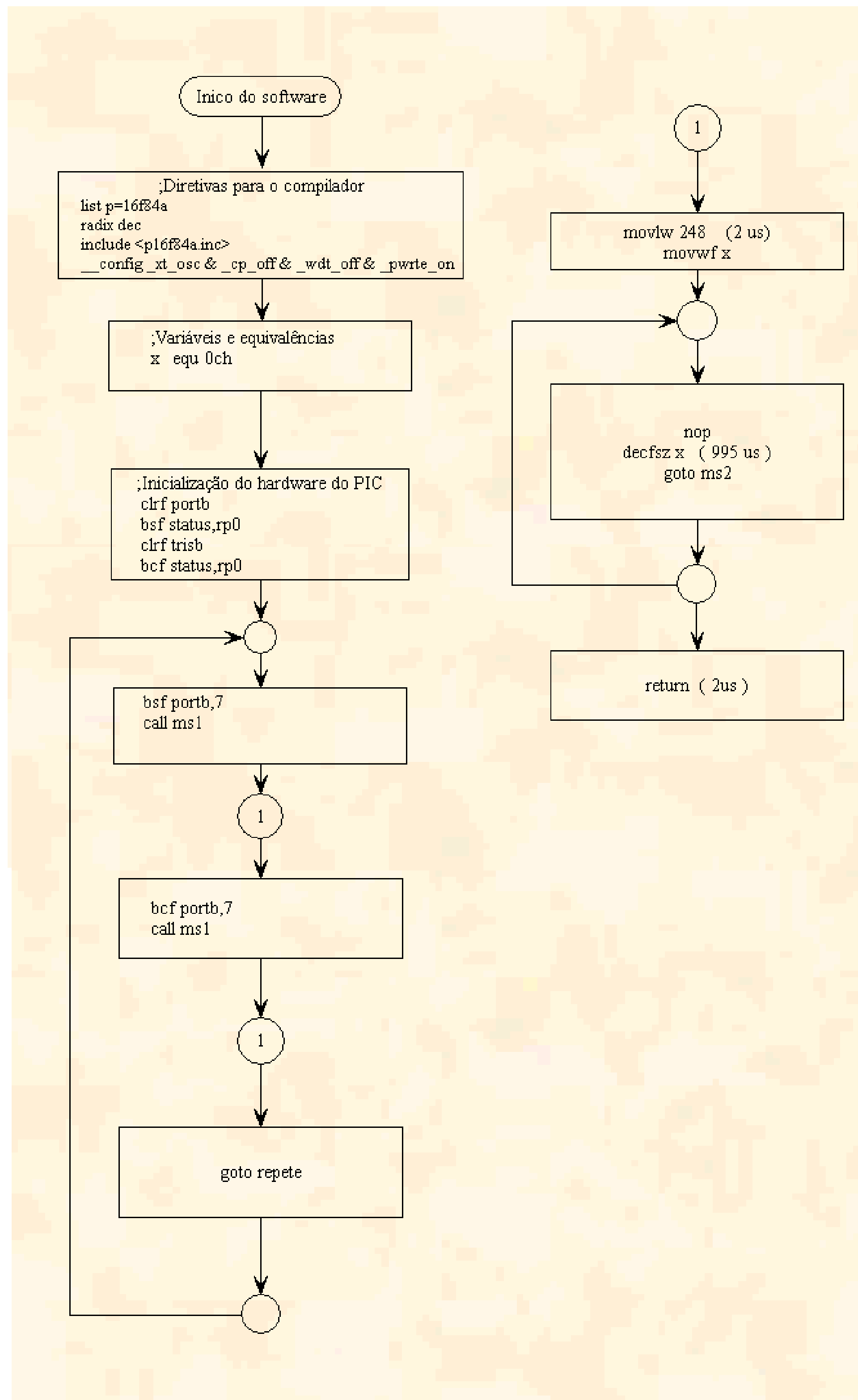
W = 01100110
k =10010010
resultado W =11110100

Rotinas de Delay

Chamos rotinas de delay, na verdade são subrotinas, aquelas que causam um atraso na programação, são rotinas que fazem o processador "perder tempo" ou "gastar um tempo". Para gastar esses tempos, de tal forma que nós os programadores possamos ter total controle sobre ele, poderemos usar dois tipos de rotinas:

- Rotina de delay por instruções
  - Apesar de qualquer delay que se for fazer, utilizar instruções, esta é uma maneira de diferenciar do delay feito com o registro timer 0.
- Rotina de delay usando o periférico Timer0 do PIC
  - Rotina que vai usar como base de tempo o contador chamado Timer0 do PIC (próxima aula)

Fluxo do programa



Passo 01

Faça todos aqueles procedimentos para criar um novo projeto. O nosso projeto vai se chamar delay, então vamos criar uma pasta delay no diretorio _PIC, fazer um código fonte chamado delay.asm, e salvar nessa pasta como na figura abaixo:

```
c:\_pic\delay\delay.asm
;#####################    TITULO DO SEU CODIGO FONTE   #######################
;#                                                                           #
;#       Rotina de delay (atraso) de 1ms utilizando apenas instrucoes        #
;#                                                                           #
;#############################################################################

;----------------------DIRETIVAS AO COMPILADOR------------------------------

        LIST      P=16F84A
        RADIX     DEC
        INCLUDE   <P16F84A.INC>
        __CONFIG  _XT_OSC & _CP_OFF & _WDT_OFF & _PWRTE_ON


;------------------VARIAVEIS E EQUIVALENCIAS--------------------------------


x       equu     0ch     ;atribui um nome a posicao de memória 0ch (do usuario)

;-------------------INICIALIZAÇÃO DO PROGRAMA--------------------------------

        clrf     portb          ; Faz todos os bits de portb = zer0
        bsf      status,rp0     ; vai para o banco 1 da RAM
        clrf     trisb          ; Faz portb todo saida
        bcf      status,rp0     ; volta ao banco 0 da RAM

;-------------------PROGRAMA PRINCIPAL----------------------------------------

repete
        bsf      portb,7        ; Faz bit 7 do portb=1
        call     ms1            ; Chama a subrotina ms1 (nosso delay de 1ms)
        bcf      portb,7        ; Faz bit 7 do portb=0
        call     ms1            ; Chama a subrotina ms1 (nosso delay de 1ms)
        goto     repete         ; Vai para o label repete

;------------------------------ SUB ROTINAS --------------------------------

;----- inicio da rotina de 1 milisegundos
ms1
        movlw    249            ; carrega x com o valor 249 (decimal)
        movwf    x
ms2                             ; Ate aqui incluindo o call gastaram-se 4 us.
        nop                     ; + 1 us
        decfsz   x              ; + 1 us (no último eh 2 e pula p/ return)
        goto     ms2            ; + 2 us, total 4us.(no último não passa aqui)
                                ; 4 us do inicio + (4 us x 248) + 3us do último
                                ; totaliza 999 us
        return                  ; + 2 us retorna da sub-rotina apos 1,001ms
                                ; aproximadamente 1 ms
;-----fim da rotina de 1 milisegundos

;---------------------------- fim do programa --------------------------------
        end
```

faça o download o delay.asm clique aqui

Note que apareceu uma novidade, desde o último código fonte que nós estudamos. É o (;) ponto e virgula, após esse caracter, tudo que estiver escrito nessa linha o compilador ignora. Usamos este recurso para fazer os comentários dos nossos códigos fontes. O código fonte está divido em algumas partes:

- Título do seu código fonte
  - Nesse espaço descrevemos para que o código fonte foi criado, podemos escrever o que quizer, como está tudo com (;) o compilador nem passa por aí.
- Diretivas ao compilador
  - Nessa primeira parte do código fonte, informamos ao compilador as diretivas que vai informar qual o PIC, sistema de numeração, os includes, o config dos fusíveis etc.
- Variáveis e equivalências
  - Nessa parte do programa, informamos ao copilador através da diretiva "equ", menemônico de equate, os textos que vão ser substituidos por números durante a compilação, no nosso exemplo, "x" vai ser substituido por 0CH, número C em hexadecimal ou 12 em decimal, esse é o primeiro endereço de memória RAM disponível ao

usuário, no PIC16F84. Em outros PIC´s esse endereço inicial pode mudar.

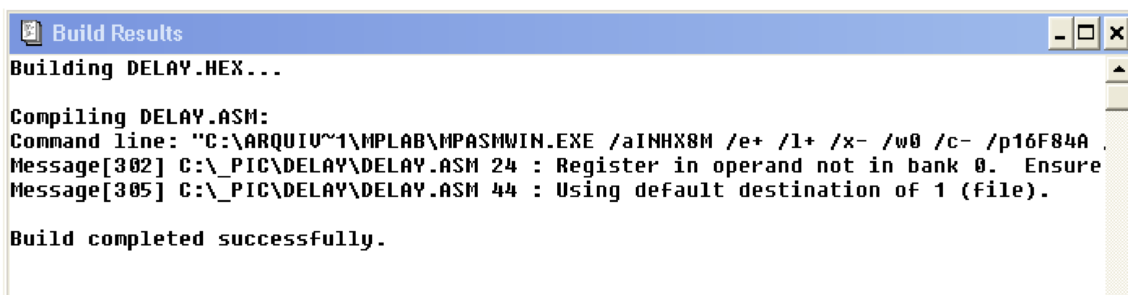
- Inicialização do Programa
  - Nessa parte fazemos a inicialização do hardware do PIC, informando que pino é entrada ou saída, quais periféricos serão habilitados etc. Fazemos a inicialização de algumas variáveis, como por exemplo a situação inicial dos ports.
- Programa principal
  - Nessa parte é que fazemos o " loop eterno", ou seja um laço de programa que fica sendo executado eternamente, aí vai a nossa rotina principal do programa.
- Sub-rotinas

Nessa parte escrevemos as diversas sub-rotinas que serão chamadas pelo programa principal.

Passo 02

Crie o projeto

1. Salve o código fonte na pasta do projeto. (delay.asm na pasta c:_pic\delay\);
2. Crie um novo projeto Project>New;
3. Clique no Project Files o [.hex] depois em Node Properties;
4. Marque em Hex Format INH8X8M , Warning level all, Case sensitive em off, clique em OK;
5. Adicione o Node, Add Node procure da pasta c:_pic\delay\delay.asm, clique em OK;
6. Clique em OK na janela do Edit Project;
7. Compile o programa clicando no ícone do funil verde, ou F10

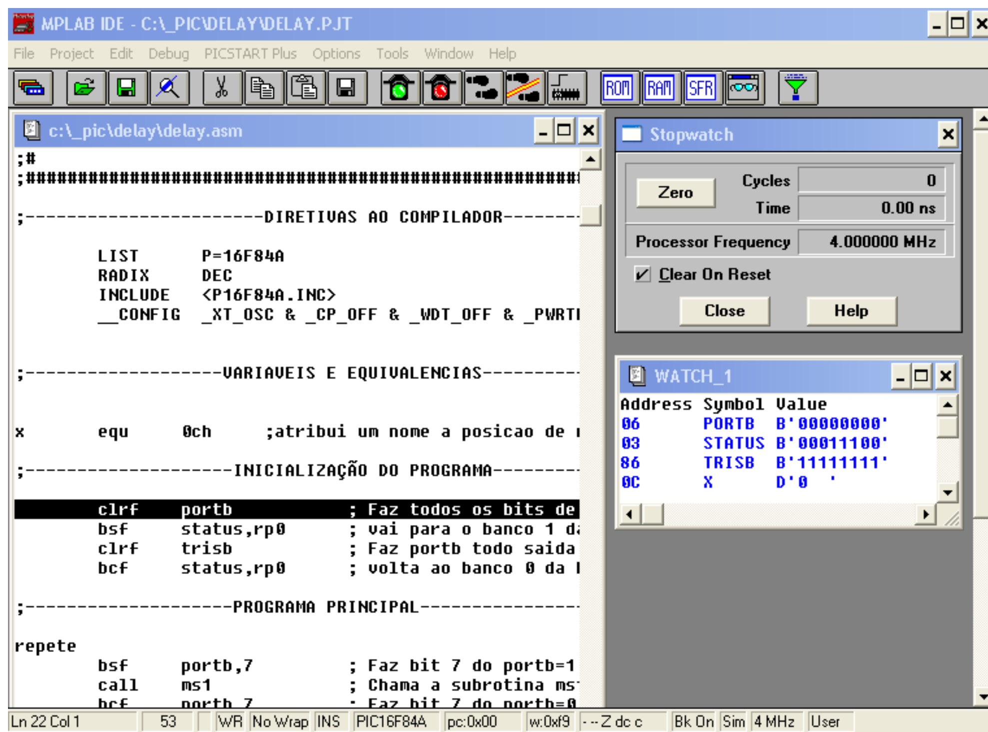


Passo 03

Feche a janela de Buid Results, adicione a janela de Stop watch, (Window >stopwatch...)

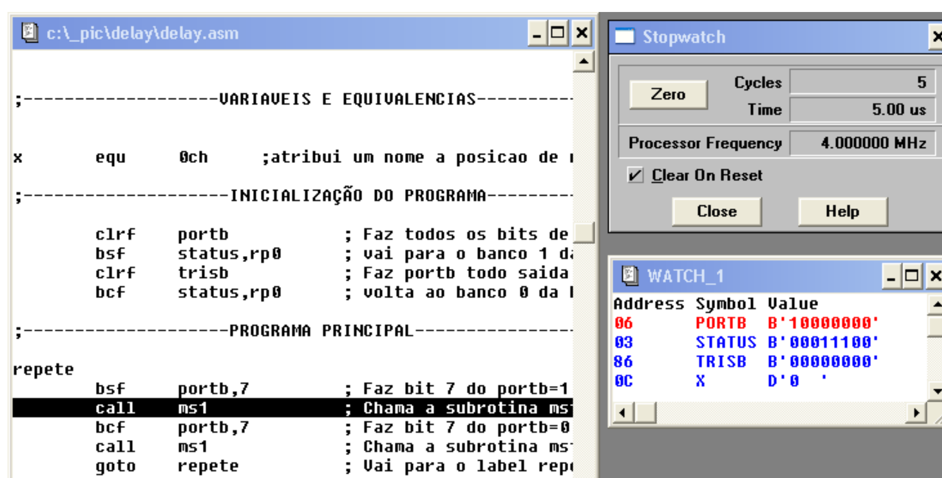
Passo 04

Adicione a janelaWatch window, (Window >watch windows>new watch windows), adicione os seguintes registros: PORTB - STATUS - TRISB em hexadecimal e X em decimal. Organize a sua área de trabalho e clique em F6(Reset), para começarmos a fazer a simulação e entender o programa. Você deve estar com tela como da figura abaixo.



Passo 05

Clique 5 vezes em F7 (executar passo a passo), até chegar no primeiro call. Até esse ponto se passaram 5 ciclos de máquina. Lembre-se que a linha tarjada ainda não foi executada, vai ser a próxima quando clicarmos em F7.



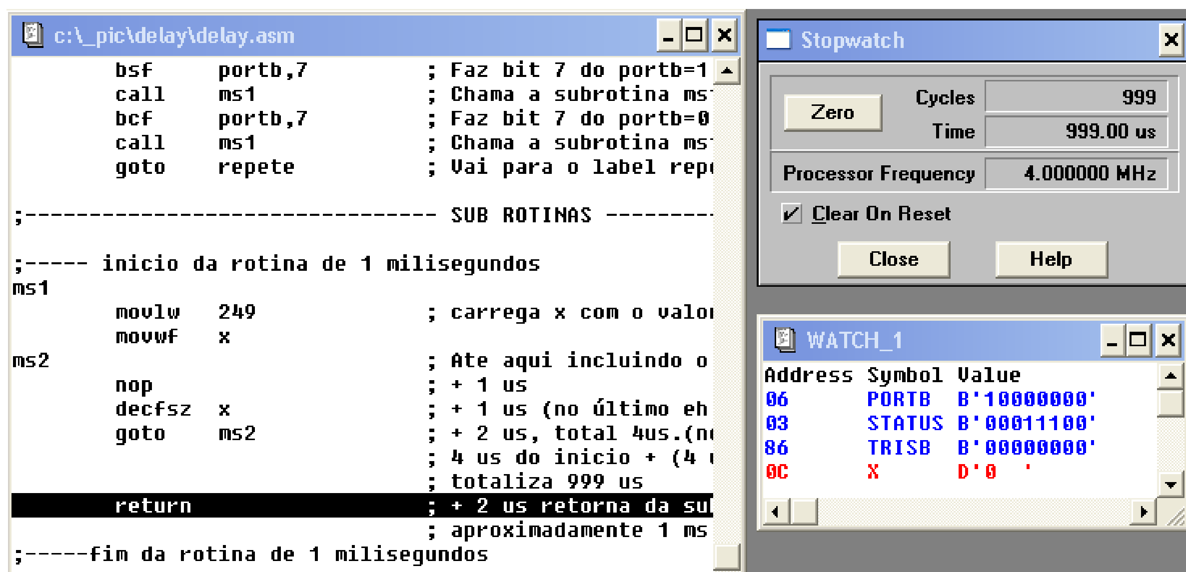
Passo 06

Clique no Zero do Stopwatch, vamos fazer isso para contar quanto tempo realmente nossa subrotina de delay vai perder.



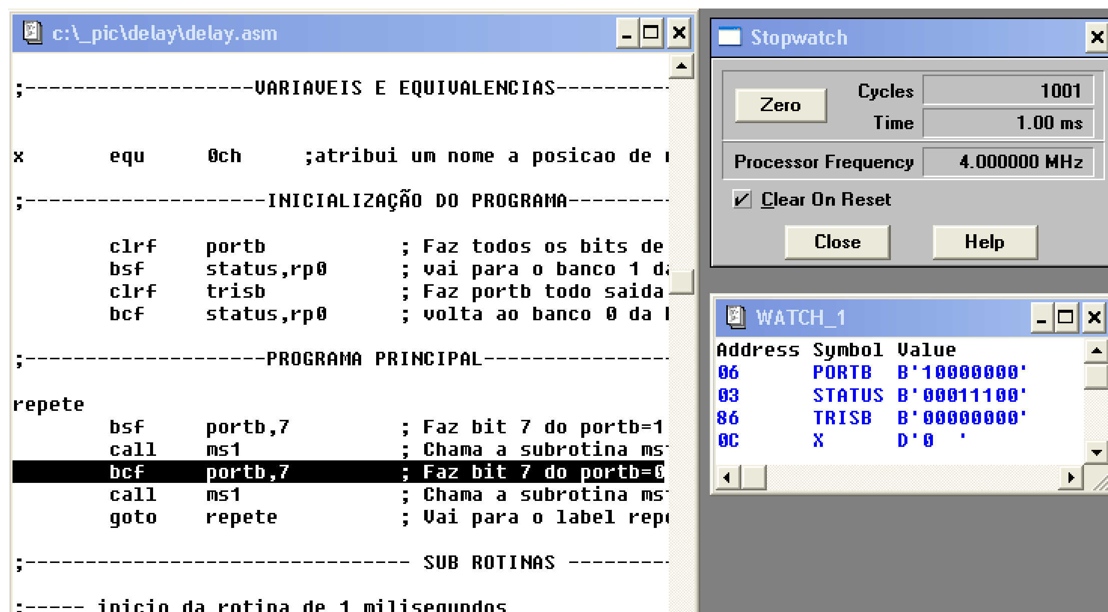
Passo 07

Clique na janela do código fonte para ativá-la. A idéia agora, é ir clicando em F7 até o programa voltar na linha de baixo da tarja, que é o [bcf portb,7]. A cada clique, faça uma verificação dos registros, na janela de watch, note que a variável x vai começar a decrementar. depois que assimilar a rotina. segure F7 pressionado para aumentar a velocidade de execução da simulação, fique de olho na variável x, quando ela chegar perto das unidades pressione F7 passo a passo de novo. Uma outra forma de é pressionar Ctrl + F9(simulação animada) e F5(Stop), quando quiser parar a simulação. Clique até chegar em return. Verifique que até esse ponto a cpu gastou 999 micro segundos, (ou ciclos de máquina), quando clicarmos F7 novamente a cpu vai gastar 2 ciclos de máquinas e voltar para o programa principal com 1001 ciclos de máquina de atraso.



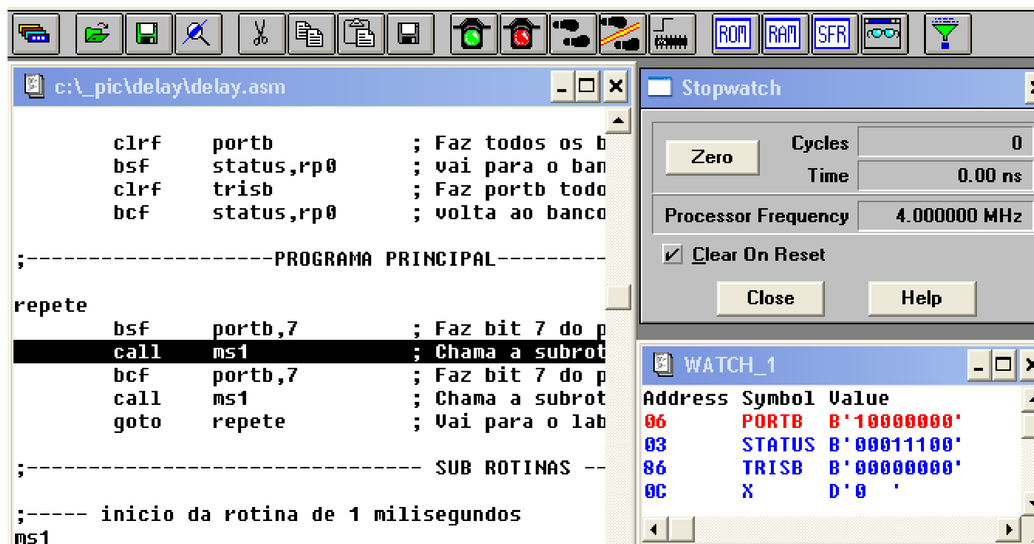
Passo 08

Entre a chamada e a subrotina gastou-se 1001 ciclos de máquina, como cada ciclo com o cristal de 4MHz que estamos usando, equivale a 1 us, isto vai dar 1,001 milisegundos, ou seja praticamente 1 ms.



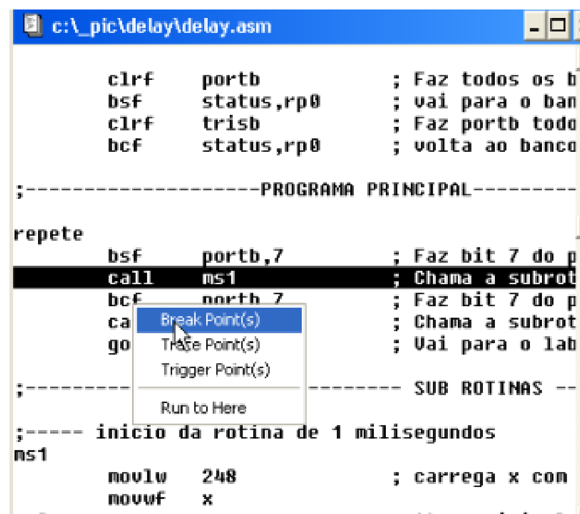
Passo 09

Uma outra forma de fazer a simulação sem a animação. consiste em usar linhas de break point no programa, isto é a gente marca uma linha, executa a simulação sem a animação, e quando chegar na linha desejada a simulação para automaticamente. vamos ver: clique em F6, vá até o primeiro call e zere o Stopwatch.



Passo 10

Clique com lado esquerdo do mouse sobre a linha abaixo do call, o bcf portb,7, é nessa linha que queremos colocar o break point, o cursor vai estra piscando, aí você clica com o lado direito do mouse, ainda sobre a linha e selecione Break point(s) clicando com o lado esquerdo.



a linha do break point vai ficar vermelha. se quiser retirar o break point repita a operação acima.

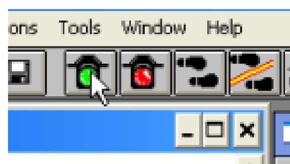
```
;------------------PROGRAMA PRINCIPAL---------

repete
          bsf       portb,7           ; Faz bit 7 do p
          call      ms1               ; Chama a subrot
          bcf       portb,7           ; Faz bit 7 do p
          call      ms1               ; Chama a subrot
          goto      repete            ; Vai para o lab

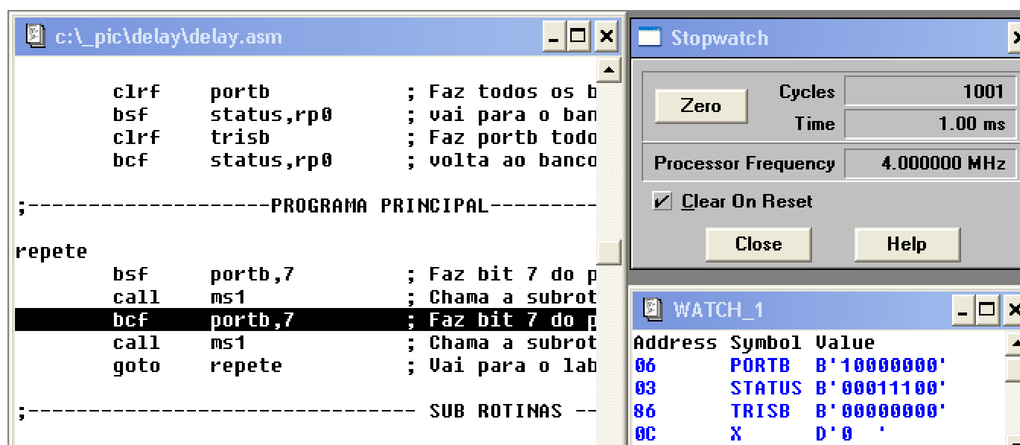
;-------------------------------- SUB ROTINAS --
```

Passo 11

Clique em F9 ou no ícone " semáforo verde "



Após um tempo a simulação para automaticamente na linha marcada pelo break point. quando maior o tempo do atraso, maior é o tempo da simulação, você vai notar quando fizer um programa para 100 ms.



Explicação da rotina de delay

Como fazer para criar rotinas de atraso?

Primeiramente temos que pensar em rotinas que ficam decrementando uma variável e executando alguns comandos que perdem tempo, como o nop, lembrando que o número máximo de uma variável no pic é 255. Na nossa Rotina fizemos a variável x=249, numa rotina que perde 4 ciclos de máquinas a cada decremento, apenas na última passagem são 3 ciclos, pois a instrução decfsz quando a variável é zero, usa dois ciclos e salta a próxima instrução que no caso é um goto (2 ciclos), então fica 1 do nop + 2 do decfsz que somam 3.

Calculando os ciclos de máquina rotina:

1. Devemos levar em consideração a chamada a subrotina o call (2 ciclos)
2. Inicializaçaõ da variável x (movlw 249 - movwf x) (2 ciclos) até aqui total de 4 ciclos
3. Como x = 249, isto dá um tempo de: { 4us(nop + decfsz x + goto ms2) X 248 = 992} até aqui total de 996 ciclos
4. Último decremento (1 ciclo do nop + 2 ciclos do decfsz, pula o goto) 3 ciclos, até aqui 999
5. Agora chegou no return (2 ciclos) totalizando 1001 ciclos de máquina.

Dependendo da aplicação essa diferença de 1 us é tolerável.

Passo 12

Tem como fazer dar exatamente 1 milisegundos? Sim é lógico, para isso no programa tinhamos que fazer x = 248, isto daria um tempo de:{ 4 us(call + movlw 248 + movwf x)} + { 4us(nop + decfsz x + goto ms2) X 247 = 988 (rotina de decrementar)} + { 3us do último decremento} = 995.

Como teremos mais 2 us do return, faltam 3us para completar 1000 ciclos, coloca-se então 3 nop antes do return. Faça a modificação, compile o programa novamente e verifique.

```
;------------------------------- SUB ROTINAS --------

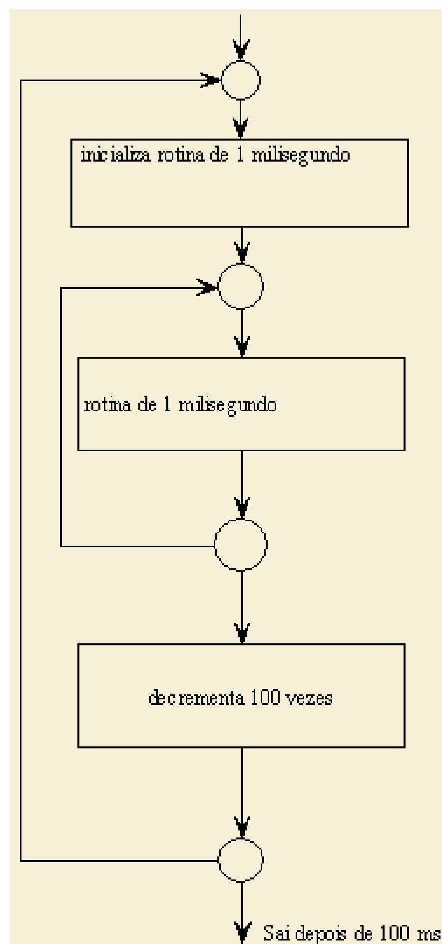
;----- inicio da rotina de 1 milisegundos
ms1
        movlw   248             ; carrega x com o valor
        movwf   x
ms2                             ; Ate aqui incluindo o
        nop                     ; + 1 us
        decfsz  x               ; + 1 us (no último eh
        goto    ms2             ; + 2 us, total 4us.(n
                                ; 4 us do inicio + (4 
                                ; totaliza 999 us
        nop
        nop
        nop
        return                  ; + 2 us retorna da sub
                                ; aproximadamente 1 ms
;-----fim da rotina de 1 milisegundos

;---------------------------- fim do programa -------
        end
```

Aumentando o tempo do atraso.

E se precisarmos de mais tempo? como fazer? por exemplo 100 ms.

Bom, aí temos que por a cabeça pra funcionar... se essa rotina que fizemos gera aproximadamente 1 ms, basta executá-la 100 vezes. fazemos então uma rotina para decrementar uma outra variável 100 vezez, e a cada decremento executar a rotina de 1 ms.



Tente elaborar essa rotina sozinho, Lembre-se que vai ter que usar uma variável auxiliar a mais, por exemplo eu usei milisegundo.

```
;#####################   TITULO DO SEU CODIGO FONTE   ######################
;#                                                                         #
;#        Rotina de delay (atraso) de 100ms utilizando apenas instrucoes   #
;#                                                                         #
;###########################################################################

;-----------------------DIRETIVAS AO COMPILADOR------------------------------

        LIST      P=16F84A
        RADIX     DEC
        INCLUDE   <P16F84A.INC>
        __CONFIG  _XT_OSC & _CP_OFF & _WDT_OFF & _PWRTE_ON


;-------------------VARIAVEIS E EQUIVALENCIAS---------------------------------


x               equ     0ch     ;variavel auxiliar no delay
milisegundo     equ     0dh     ;variavel auxiliar no delay

;--------------------INICIALIZAÇÃO DO PROGRAMA---------------------------------

        clrf    portb           ; Faz todos os bits de portb = zer0
        bsf     status,rp0      ; vai para o banco 1 da RAM
        clrf    trisb           ; Faz portb todo saida
        bcf     status,rp0      ; volta ao banco 0 da RAM

;--------------------PROGRAMA PRINCIPAL-----------------------------------------

repete
        bsf     portb,7         ; Faz bit 7 do portb=1
        call    ms100           ; Chama a subrotina ms1 (nosso delay de 1ms)
        bcf     portb,7         ; Faz bit 7 do portb=0
        call    ms100           ; Chama a subrotina ms1 (nosso delay de 1ms)
        goto    repete          ; Vai para o label repete

;-------------------------------- SUB ROTINAS ---------------------------------

;----- inicio da rotina de 100 milisegundos

ms100
        movlw   100
        movwf   milisegundo     ; Até aqui incluindo o call gastaram-se 4 us.

ms1
        movlw   249             ; carrega x com o valor 249 (decimal)
        movwf   x               ; 2 us ( 1 do movlw e 1 do movwf )
ms2
        nop                     ; + 1 us
        decfsz  x               ; + 1 us (no último eh 2 e pula p/ return)
        goto    ms2             ; + 2 us, total 4us.(no último não passa aqui)
                                ; (4 us x 249)-1  totaliza 995 us
        decfsz  milisegundo     ; +1 us (na última pasagem 2 us)
        goto    ms1             ; +2 us (na última passagem pula)
                                ; Total antes do return: 4us (inicio) +100 X 1000 us
                                ; (1 do movlw 249 + 1 do movwf x + 995 da rotina ms1
                                ; + 1 do decfsz + 2 do goto)- 1 da última passagem
                                ; Total = 100003 us
        return                  ; + 2 us retorna da sub-rotina apos 100005 us
                                ; aproximadamente 100 ms
;-----fim da rotina de 100 milisegundos

;----------------------------- fim do programa ---------------------------------
        end
```

Rotinas de Delay com Timer0

O Periférico Timer0, é um contador interno do PIC, pode funcionar com estímulo externo ou fazer as contagens internamente a partir do oscilador, possui um prescaler que vai até 1:256. O registro onde fica armazenado as contagens é o TMR0, menemônico dado pela Microchip que vem no arquivo de include, na realidade é o endereço de RAM 01H, o registro especial que controla o Timer0 é Option, o menemônico é *option_reg,* o endereço da RAM é o 81H, está no banco1, da mesma forma que o TRISB. Em aula anterior já vimos esse registro, lembrando que um registro de controle possui 8 bits, e nem todos são usados somente para o Timer0. Um outro registro de controle que vamos usar o INTCON.

Registro OPTION_REG

Propriedade	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Bit	7	6	5	4	3	2	1	0
Nome	/RBPU	INTDG	T0CS	T0SE	PSA	PS2	PS1	PS0

Vamos comentar apenas os bits que dizem respeito ao Timer0: o T0CS (Timer Zero Chip Select) vai atribuir a fonte de clock para o contador, em 1 clock é externo, por exemplo podemos usar para contar número de voltas de um eixo, se houver um sensor de voltas é lógico. se em zero o clock é interno, vai usar os ciclos de máquina como base.

T0CS - bit de seleção de fonte de clock do timer 0
1 - Transição no pino RA4/T0CKI
0 - Clock interno (CLKOUT = Fosc/4)

O T0SE (Timer Zero Select Edge), caso o Timer0 seja selecionado para clock externo,(Pino RA4/T0CKI Timer Zero Clock Input), esse bit indica se o clock ativa o timer na rampa de subida ou descida do sinal.

T0SE - bit de seleção de como incrementará o Timer 0
1 - Na descida do sinal no pino RA4/T0CKI
0 - Na subida do sinal no pino RA4/T0CKI

O PSA (Prescales Select Assignment bit), Vai atribuir se o prescaler vai atuar no Timer0 ou no Watch Dog, como só existe um prescaler no PIC16F84, temos que escolher em qual periférico vamos usá-lo.

PSA - Bit de atribuição do Prescaler
1 - Prescaler atribuído ao Watch Dog
0 - Prescaler atribuído ao TMR0

De PS0 a PS2 (Prescaler rate Select bit), conforme o valor dos bits, tabela abaixo, termos uma taxa para o prescaler. O detalhe importante que não são iguais para os dois periféricos.

PS2, PS1 e PS0 - Ajustam a taxa de divisão do Prescaler Veja a tabela na página seguinte.				
PS2	PS1	PSO	Divisão Timer 0	Divisão Watch Dog
0	0	0	1/2	1/1
0	0	1	1/4	1/2
0	1	0	1/8	1/4
0	1	1	1/16	1/8
1	0	0	1/32	1/16
1	0	1	1/64	1/32
1	1	0	1/128	1/64
1	1	1	1/256	1/128

Registro INTCON

Propriedade	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	X
Bit	7	6	5	4	3	2	1	0
Nome	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

Nesse registro de controle das interrupções, vamos usar apenas um bit para nossa aplicação, é o T0IF, (Timer Zero Interrupt Flag). Esse bit fica monitorando a posição de RAM TMR0, quando houver estouro do timer ela vai pra nível lógico 1. Detalhe importante: No reset ela está em zero, isso é lógico pois o processo está no começo e ainda não houve overflow, mas a partir do primeiro estouro o flag, isto é, o bit vai pra 1, e fica em 1 até que nós via nosso software coloquemos em zero novamente.

T0IF - Bit Sinaliza interrupção pelo Overflow do TMR0
1 - Ocorreu overflow no TMR0
0 - Não ocorreu overflow

Contando um tempo com TMR0

No TMR0, como é uma posição de RAM, podemos ler e escrever à vontade, então a contagem não precisa ser iniciada em zero, posso ajustar o valor inicial. A idéia para se fazer uma contagem de tempo é a seguinte:

- Vamos supor que o prescaler é 1:1, espera aí! veja a tabela. Não tem 1:1 certo? Como pode acontecer um prescaler 1:1 então? Para isso acontecer basta selecionar o prescaler para o Watch Dog. Ai tanto o watch dog como timer0 estarão com prescaler 1:1. (por isso aquela diferença na tabela). Continuando... Prescaler 1:1 significa que a cada 1 us, (com cristal de 4 MHz), o TMR0 é incrementado de uma unidade.
  - Quero contar um tempo de 100 us. Primeiro faço as contas de qual será o número inicial do TMR0, para que quando houver overflow, (256 já é o over flow pois o registrador é de 8 bits), tenha passado os 100 us Essa é fácil, se overflow acontece com 256, então o número inicial é 256-100 = 156.
  - Primeiro coloco o valor 156 em TMR0, depois faço o flag ficar zero, depois fico testando o flag, quando o flag for pra 1, bingo! se passaram exatamente 100 us.

A coisa começa a complicar um pouquinho, quando queremos medir tempo maiores.Usando o prescaler podemos aumentar o tempo de incremento do TMR0, mas sempre ele vai ter 8 bits, ou seja conta de zero a 255 e em 256 dá o over flow e volta a contar de zero, se cada ciclo de máquina for de 1 us, então teremos as seguintes situações possíveis de prescaler:

- 1:1

Ciclos de máquina	0	1	2	--	254	255	256	257
TMR0	0	1	2	--	254	255	0	1
Tempo	0	1us	2us	--	254us	255us	256us	257us
T0IF	0	0	0	0	0	0	1	1

- 1:2

Ciclos de máquina	0	2	4	--	508	510	512	514
TMR0	0	1	2	--	254	255	0	1
Tempo	0	2us	4us	--	508 us	510us	512us	514us
T0IF	0	0	0	0	0	0	1	1

- 1:4

Ciclos de máquina	0	4	8	--	1016	1020	1024	1028
TMR0	0	1	2	--	254	255	0	1
Tempo	0	4us	8us	--	1.016us	1.020us	1.024us	1.028us
T0IF	0	0	0	0	0	0	1	1

- 1:8

Ciclos de máquina	0	8	16	--	2032	2040	2048	2056
TMR0	0	1	2	--	254	255	0	1
Tempo	0us	8us	16us	--	2.032us	2.040us	2.048us	2.056us
T0IF	0	0	0	0	0	0	1	1

- 1:16

Ciclos de máquina	0	16	32	--	4064	4080	4096	4112
TMR0	0	1	2	--	254	255	0	1
Tempo	0us	16us	32us	--	4.064us	4.080us	4.096us	4.112us
T0IF	0	0	0	0	0	0	1	1

- 1:32

Ciclos de máquina	0	32	64	--	8128	8160	8192	8224
TMR0	0	1	2	--	254	255	0	1
Tempo	0us	32us	64us	--	8.128us	8.160us	8.192us	8.224us
T0IF	0	0	0	0	0	0	1	1

- 1:64

Ciclos de máquina	0	64	128	--	16256	16320	16384	16448
TMR0	0	1	2	--	254	255	0	1
Tempo	0us	64us	128us	--	16.256us	16.320us	16.384us	16.448us
T0IF	0	0	0	0	0	0	1	1

- 1:128

Ciclos de máquina	0	128	256	--	32512	32640	32768	32896
TMR0	0	1	2	--	254	255	0	1
Tempo	0us	128us	256us	--	32.512us	32.640us	32.768	32.896us
T0IF	0	0	0	0	0	0	1	1

- 1:256

Ciclos de máquina	0	256	512	--	65024	65280	65536	65792
TMR0	0	1	2	--	254	255	0	1
Tempo	0us	256us	512us	--	65.024us	65.280us	65.536us	65.792us
T0IF	0	0	0	0	0	0	1	1

No caso do maior prescaler, teremos o estouro em 65536 us, aproximadamente 65,54 milisegundos. Acontece o seguinte quando usamos o prescaler: Como são 255 incrementos, entre um incremento e outro se passam alguns ciclos, se o tempo que queremos está nesse meio, ou seja, não é um multiplo exato do incremento, temos que escolher um valor ligeiramente abaixo, e depois usar uma rotina de delay com instrução para ajustar o tempo exato. Com alguns cálculos e o auxílio do Simulador do MPlab, o Stopwatch, é possível ajustar qualquer valor exato de tempo.

E se o tempo que queremos for ainda maior que 65,536 ms?

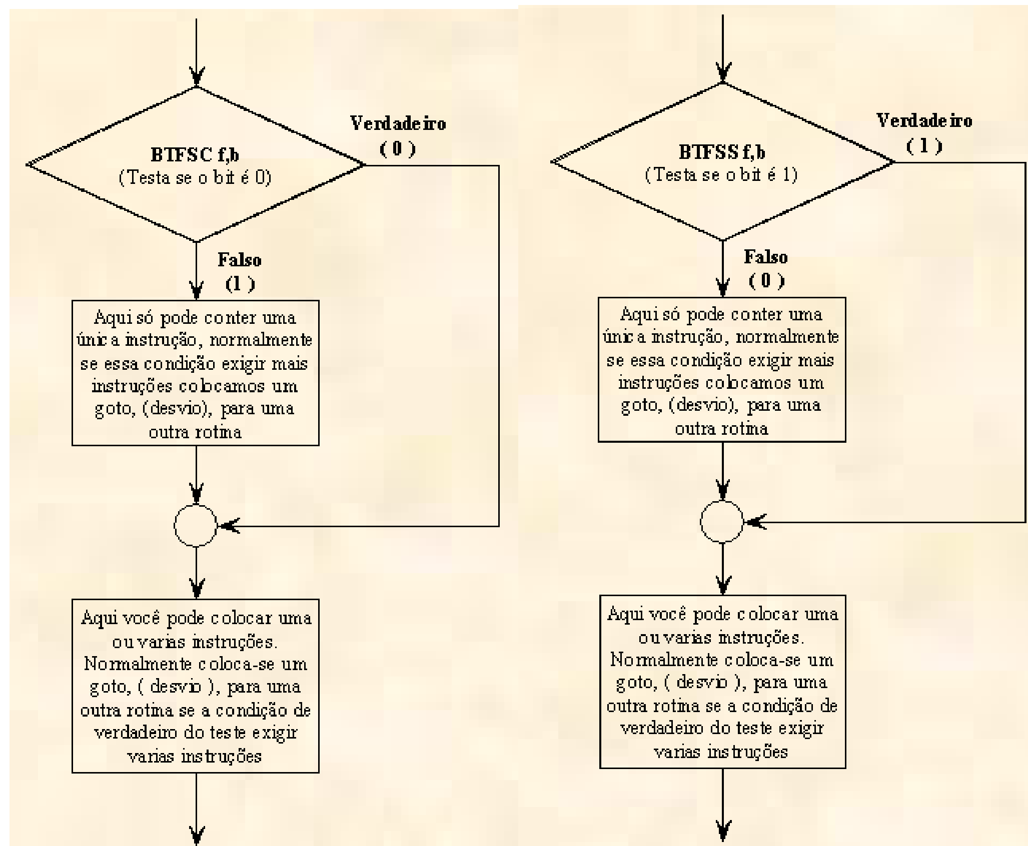
Nesse caso temos que montar uma rotina usando uma variável auxiliar, que conta o número de estouros de TMR0. Por exemplo: 1 segundo é um milhão de microsegundos, 1.000.000/65536=15,258... então com 15 estouros de TMR0 mais um pequeno ajuste chegamos no 1 segundo exato.

Aplicação:

Delay de 100 ms com Timer0

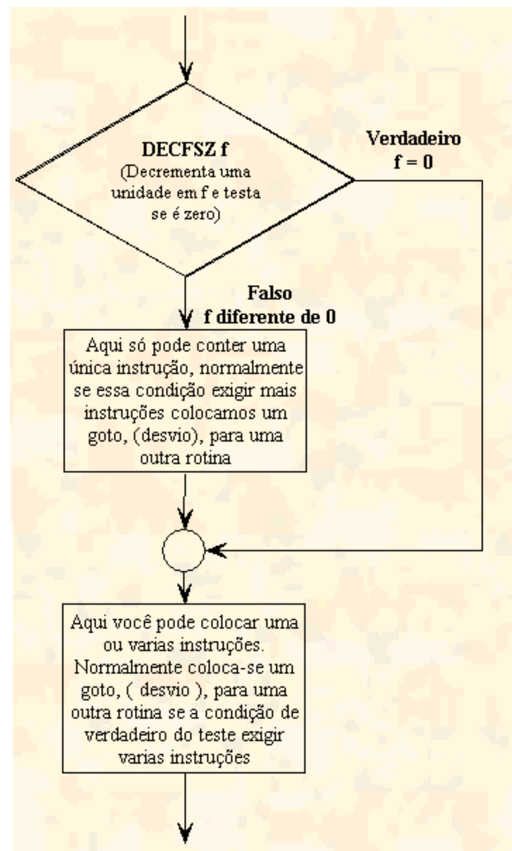
Vamos usar nesse programa uma instrução que testa um bit, no PIC há apenas duas instruções para se testar um único bit, é a BTFSS (testa se o bit é 1) e a BTFSC (testa se o bit é zero). A instrução BTFSS testa se o bit é Set (1), então se o bit for mesmo 1 significa que o teste resultou verdadeiro, nesse caso a ação da instrução é pular a próxima instrução. A instrução BTFSC testa se o bit é Set (0), então se o bit for mesmo 0significa que o teste resultou verdadeiro, nesse caso a ação da instrução é pular a próxima instrução

Instrução *BTFSS f,b* e *BTFSC f,b*



Vamos usar também uma outra instrução que decrementa uma unidade de um registro, e testa se chegou a zero, quando a situação zero é verdadeira a ação é pular a próxima instrução, assim como as instruções de testar bit.

Instrução *DECFSZ f*



Passo 01

Faça todos aqueles procedimentos para criar um novo projeto. O nosso projeto vai se chamar delayt0, então vamos criar uma pasta delayt0 no diretorio _PIC, fazer um código fonte chamado delayt0.asm, e salvar nessa pasta . clique aqui para o download do arquivo

c:_pic\delayt0\delayt0.asm

```
;##################################################################
;#                                                                #
;#        Rotina de delay (atraso) utilizando Timer0 (TMR0)       #
;#                                                                #
;# Este fonte utilizara o timer0 para gerar um delay de 100 ms para  #
;#  que na saida Rb7 seja possivel visualizar o led1 piscar.      #
;# Neste exemplo o valor do prescaler foi 1:256                   #
;#                                                                #
;# CURSO ON LINE DE PIC    www.edutecbauru.com.br                 #
;#                                                                #
;##################################################################

;----------------------DIRETIVAS AO COMPILADOR----------------------------

        LIST    P=16F84A
        RADIX   DEC
        INCLUDE <P16F84A.INC>
        __CONFIG _XT_OSC & _CP_OFF & _WDT_OFF & _PWRTE_ON


;------------------VARIAVEIS E EQUIVALENCIAS--------------------------------

y       equ     0ch     ;atribui um nome a posicao de memória 0ch (do usuario)
fino    equ     0dh     ; idem a posicao 0dh
led1    equ     7       ;atribui um nome ao bit 7 do port B, para facilitar
                        ;na programacao

;-------------------INICIALIZAÇÃO DO PROGRAMA---------------------------------

        bsf     status,rp0      ;vai para o banco 1
        bcf     option_reg,t0cs ;faz timer 0 direcionado para o clock interno
        bcf     option_reg,psa  ;prescaler direcionado para o timer0
                                ;e taxa de 1:256 (pois esse é o default no reset)
        bcf     trisb,7         ;faz rb7 como saida
        bcf     status,rp0      ;volta para banco 0

;-------------------PROGRAMA PRINCIPAL-----------------------------------------
loop
        bcf     portb,led1      ;faz Rb7=0, acende o led1
        call    ms100           ;gasta 100 ms
        bsf     portb,led1      ;faz rb7=1, apaga o led1
        call    ms100           ;gasta 100 ms
        goto    loop            ;vai para o loop e repete as mesmas instrucoes

;-------------------------------- SUB ROTINAS ---------------------------------

;----- inicio da rotina de 100 milisegundos usando timer0

ms100
        bcf     intcon,t0if     ;garante o flag t0if em 0
        movlw   2               ;o timer 0 ira contar ate o overflow 2 vezes
        movwf   y
ms100a
        movlw   61              ; inicia o timer0 a partir do valor 61 (decimal)
        movwf   tmr0            ; restando para 256 mais 195 incrementos do
                                ; timer0 195 x 256 us = 49920 us ou 49,9 ms
verifica
        btfss   intcon,t0if     ; testa o flag t0if, se 0 segue a rotina
        goto    verifica        ; vai para verifica e testa o flag t0if
        bcf     intcon,t0if     ; se chegou aqui, ocorreu overflow no timer 0.
                                ; zera o flag
                                ; t0if
        decfsz  y               ; Decrementa y em 1 e testa se y=0. Com esta
                                ; instrucao fara o timer 0 contar duas vezes,
                                ; 2 x 49920 us = 99840 us ou 99,840 ms aproximado

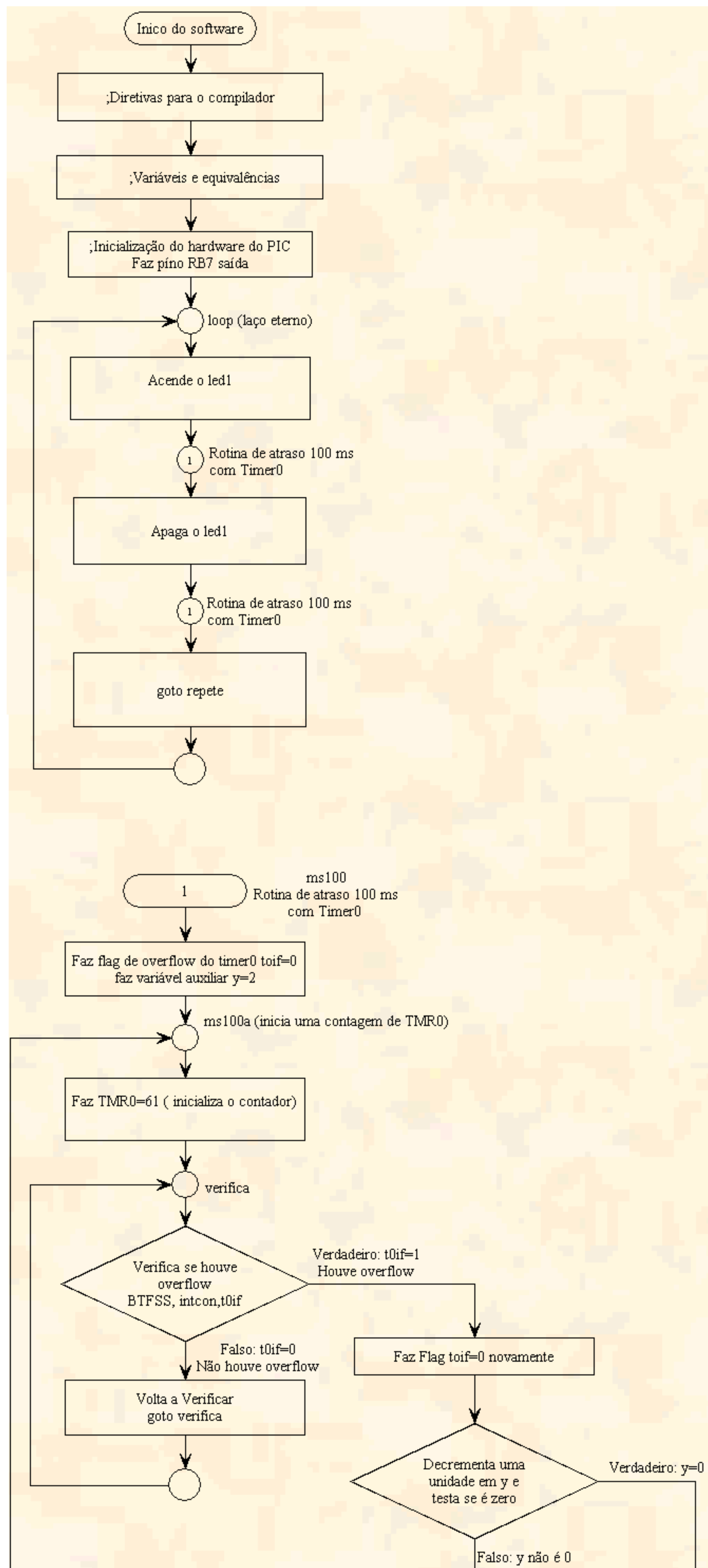
        goto    ms100a          ; vai para ms100a  carrega timer 0, y nao chegou
                                ; a zero movlw  43 ajuste para dar o tempo de
                                ; 100 ms (100000 ciclos de máquina), impossivel
                                ; conseguir um tempo exato somente com o timer 0
        movlw   43
        movwf   fino
ajuste
        decfsz  fino            ; decremeta o endereco fino (0dh) e testa se
                                ; chegou a zero.
        goto    ajuste          ; não chegou a zero, volta ao ajuste
        nop
        nop                     ; adicionado mais dois nop (2 ciclos) para
                                ; acertar o tempo
        return                  ; retorna da sub-rotina, apos 100 ms


;----- fim da rotina de 100 milisegundos usando timer0

;------------------------------ fim do programa ----------------------------------
        end
```

Explicação do programa

FLUXOGRAMA DO PROGRAMA


Inico do software
;Diretivas para o compilador
;Variáveis e equivalências
;Inicialização do hardware do PIC
Faz píno RB7 saída
loop (laço eterno)
Acende o led1
Rotina de atraso 100 ms
com Timer0
Apaga o led1
Rotina de atraso 100 ms
com Timer0
goto repete
1
ms100
Rotina de atraso 100 ms
com Timer0
Faz flag de overflow do timer0 toif=0
faz variável auxiliar y=2
ms100a (inicia uma contagem de TMR0)
Faz TMR0=61 (inicializa o contador)
verifica
Verifica se houve
overflow
BTFSS, intcon,t0if
Verdadeiro: t0if=1
Houve overflow
Falso: t0if=0
Não houve overflow
Faz Flag toif=0 novamente
Volta a Verificar
goto verifica
Decrementa uma
unidade em y e
testa se é zero
Verdadeiro: y=0
Falso: y não é 0

Vamos às explicações. Procurei fazer o arquivo fonte bastante comentado, para facilitar o entendimento. Devemos sempre comentar os nossos códigos fontes, pois quando fazemos o programa estamos pensando exatamente como queremos as rotinas, mas depois de alguns meses quando você pegar o seu código fonte para estudar, ou mudar alguma coisa, bau bau.... Dificilmente você vai se lembrar como fez as rotinas.

No início do software é sempre a mesma coisa, as diretivas ao compilador.

Nas variáveis e equivalência vamos colocar nomes nas variáveis auxiliares do nosso software, dar nomes aos atuadores, no nosso exemplo usamos o nome led1 para representar o pino RB7, na verdade apenas o compilador vai substituir o texto led1 pelo número 7 toda vez que encontrá-lo. Usamos duas variáveis auxiliares para a rotina de tempo, o y e o fino. o nome fino foi escolhido para representar um " ajuste fino", escolhemos nomes de variáveis, que lembre alguma coisa do que ela está executando.

Na inicialização do programa, é o momento de inicializar o hardware do PIC, no nosso caso fizemos o timer0 direcionado para o clock interno, com um prescaler de 1:256, fizemos o pino RB7 como saída. Fizemos isto pois a nossa aplicação está usando o timer0 para um temporizador, e o pino RB7 para atuar em um led.

O programa principal é igual ao anterior, um "pisca led", só que mais rápido, ele pisca em intervalos de 100 ms. Então uma instrução acende o led, chama a rotina de delay, apaga o led, chama a rotina de delay, acende o led de novo e ... e assim fica eternamente.

Nosso Alvo é realmente a subrotina, então vamos tentar entender. O tempo que eu quero é 100ms, ou seja, 100.000 micro segundos, no nosso caso cem mil ciclos de máquina. Na maior situação de contagem do Timer com o prescaler de 1:256 é de 65.536 ciclos de máquina na hora que o flag de over flow vai pra 1.

- Prescaler 1:256

Ciclos de máquina	0	256	512	--	65024	65280	65536	65792
TMR0	0	1	2	--	254	255	0	1
Tempo	0us	256us	512us	--	65.024us	65.280us	65.536us	65.792us
T0IF	0	0	0	0	0	0	1	1

Então uma única contagem não seria suficiente, mas duas contagens inteiras vai ser muito, vai dar 131.072 ciclos de máquinas, aproximadamente 131 ms. Fazemos então um cálculo para iniciar a contagem num número X de tal forma que no segundo over flow cheguemos bem perto dos 100.000 ciclos. Pensando.... então tenho que dividir os cem mil em duas contagens... 100.000 / 2 = 50.000. Pronto, já temos um dado.... cada contagem tem que ser de aproximadamente cinquenta mil, eu estou falando em aproximadamente, pois os incrementos no registro timer0 se dão a cada 256 ciclos, então posso ter um erro máximo de até 255 unidades pra menos, ou pra mais do que eu quero exatamente. Vamos lá qual número que somado a 50.000 vai da 65.536, qué o número que vai setar o flag t0if? Fácil... 65.536 - 50.000 = 15.536. esse é o número inicial de ciclos de máquinas que teriamos de colocar em TMR0. Mas TMR0 vai de zero a 255, que número X inicial do TMR0 é 15.536 ciclos de máquina? Está vendo a tabela alí em cima? a cada 256 ciclos acrescenta 1 no TMR0 então teremos a seguinte situação:

15.536 / 256 = 60,6875

Ciclos de máquina	0	256	512	--	--	15.360	15.616	--
TMR0	0	1	2	--	--	60	61	--

Tempo	0us	256us	512us	--	--	15.360us	15.616us	--
T0IF	0	0	0	0	0	0	0	--

Tenho que escolher um X para o TMR0, 60 ou 61 ?? Qual escolher? Pensando de novo...se eu escolher o 60, até 256 que é onde vai setar o flag de over flow vão se passar (256 - 60)x256 ciclos = (196)x256 = 50.176 ciclos. Como vou fazer duas contagens o número total de ciclos vão ser 50.176 x 2 = 100.352 ciclos de máquinas, isso é maior que os cem mil que eu quero. como tempo não dá pra voltar, é preferível escolher o X = 61, que vai dar um tempo menor, (256 - 61) x 256 = 49.920, isso vezes duas contagens dá 99.840, o que é menor que cem mil, mas com um pequeno ajuste fino de atraso podemos chegar exatamente aos cem mil.

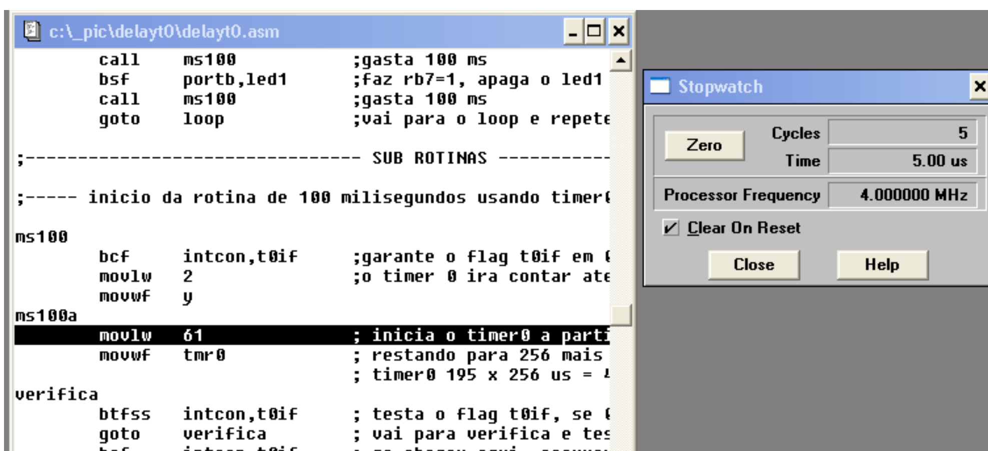
Formulinha útil: X =Inteiro de {256 - [NCOF - NCPC] / 256} + 1

NCOF = Número do ciclo de máquina que acontece o over flow (depende do prescaler)

NCPC = Número de ciclos de máquinas por contagem do TMRO

Passo 02

Agora temos que calcular o ajuste fino: temos dois ciclos do call, mais um do bcf intcon,,toif, mais um do movlw 2 e mais um do movwf y total de 5 ciclos. Use a simulação pra ajudar. Lembre-se que a linha com a tarja preta ainda não foi executada. então os 5 ciclos são até o movwf y. Ah não se esqueça de zerar o stopwatch na linha do call no programa principal.



Passo 03

Coloque um break point na linha movlw 43, é nesse ponto que o programa vai chegar após as duas contagens de TMR0.

```
ms100a
        movlw   61              ; inicia o timer0 a parti
        movwf   tmr0            ; restando para 256 mais
                                ; timer0 195 x 256 us = 4
verifica
        btfss   intcon,t0if     ; testa o flag t0if, se 6
        goto    verifica        ; vai para verifica e tes
        bcf     intcon,t0if     ; se chegou aqui, ocorreu
                                ; zera o flag
                                ; t0if
        decfsz  y               ; Decrementa y em 1 e tes
                                ; instrucao fara o timer
                                ; 2 x 49920 us = 99840 us

        goto    ms100a          ; vai para ms100a  carreg
                                ; a zero movlw  43 ajuste
                                ; 100 ms (100000 ciclos c
                                ; conseguir um tempo exat
        movlw   43
        movwf   fino
ajuste
        decfsz  fino            ; decremeta o endereco fi
                                ; chegou a zero.
        goto    ajuste          ; não chegou a zero, volt
```

Stopwatch — Zero — Cycles: 5 — Time: 5.00 us — Processor Frequency: 4.000000 MHz — Clear On Reset — Close — Help

Passo 04

Clique F9 ou no ícone do semáforo verde. espere parar....

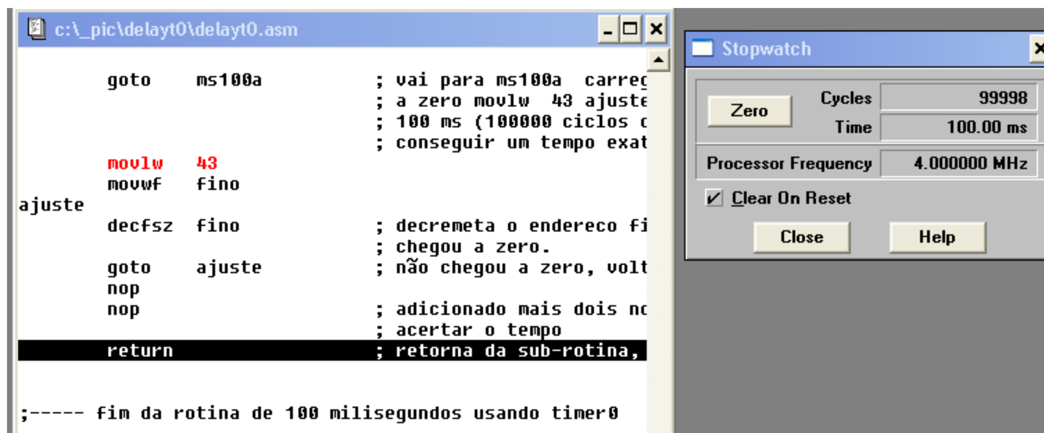
```
        movwf   y
ms100a
        movlw   61              ; inicia o timer0 a parti
        movwf   tmr0            ; restando para 256 mais
                                ; timer0 195 x 256 us = 4
verifica
        btfss   intcon,t0if     ; testa o flag t0if, se 6
        goto    verifica        ; vai para verifica e tes
        bcf     intcon,t0if     ; se chegou aqui, ocorreu
                                ; zera o flag
                                ; t0if
        decfsz  y               ; Decrementa y em 1 e tes
                                ; instrucao fara o timer
                                ; 2 x 49920 us = 99840 us

        goto    ms100a          ; vai para ms100a  carreg
                                ; a zero movlw  43 ajuste
                                ; 100 ms (100000 ciclos c
                                ; conseguir um tempo exat

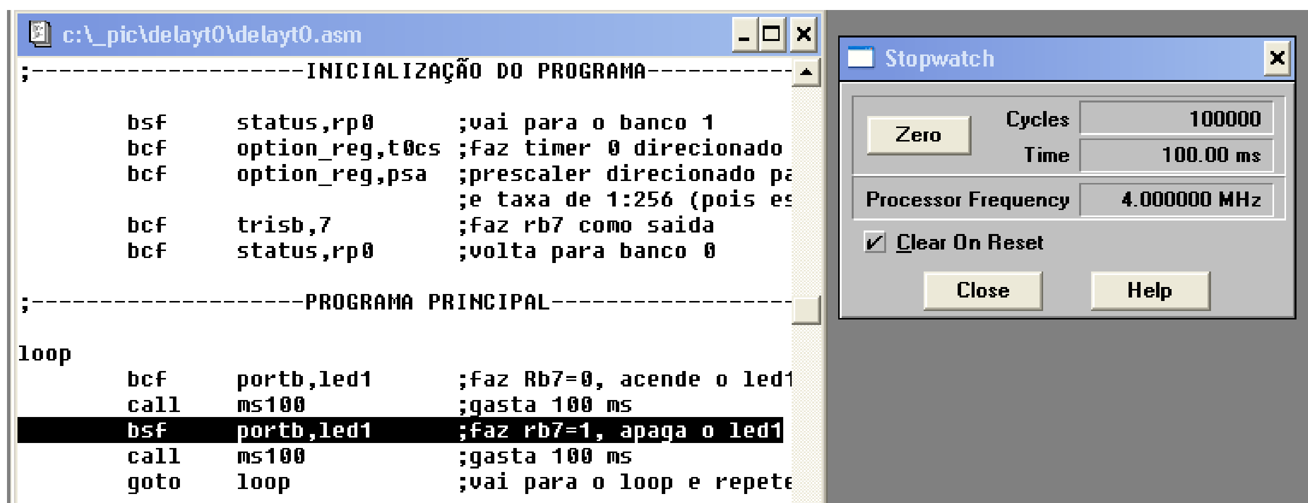
        movlw   43
        movwf   fino
ajuste
        decfsz  fino            ; decremeta o endereco fi
```

Stopwatch — Zero — Cycles: 99866 — Time: 99.87 ms — Processor Frequency: 4.000000 MHz — Clear On Reset — Close — Help

Chegou aí com 99.866 ciclos de máquinas. Porque que não deu os 99.840 que calculamos? No cálculo a gente não contou os ciclos de máquinas das instruções... mas com o recurso do simulador isso dispensável, basta saber que sempre vai dar um pouco a mais do que vc calculou. Posso até afirmar que é quase impossível fazer um cálculo exato, pois as vezes o flag t0if pode setar logo após a verificação, e como tem o goto e depois o teste, isso fica difícil de prever. Mas com o auxílio da simulação não tem erro. Encontramos 99.866 ciclos, então para 100.000 faltam 134 ciclos. Vamos fazer uma rotina de atraso decrementando uma variável até chegar a zero. essa rotina perde aproximadamente 3 ciclos por decremento 1 do decfsz e 2 do goto, só na entrada e na saída da rotina que muda um pouco, 1 do movlw k mais 1 do movwf f, na entrada da subrotina e apenas 2 na saída pois pula o goto e o decfsz f quando é verdadeiro gasta 2 ciclos. Mas fazendo a base de 3 teremos 134 / 3 = 44,66. Escolho sempre o menor pra fazer o teste. então vai ficar 2 + 44x3 -1 = 133. então somando mais 2 do return vou ter 135. oque vai dar um total de 100.001 ciclos. Pra fazer 100.000 exatos, então escolhemos 43, ai fica 2 + 43x3 -1 = 130 mais 2 nop 132, mais dois do return 134.. somando exatamente 100.000.

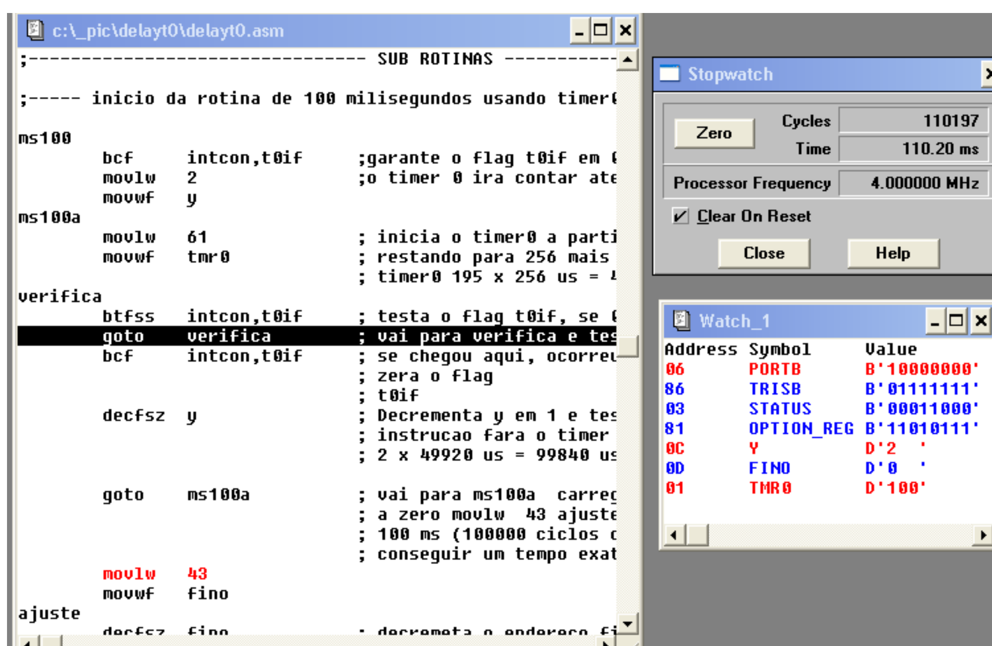


Volta da subrotina exatamente depois de 100.000 microsegundos = 100 ms.



Passo 05

Adicione a janela de watch window e visualise todos os registros importantes. faça inumeras simulações observando cada detalhe dos registros.



Fazendo um circuito para piscar um LED.

Com a rotina que acabamos de ver, já é possível fazer uma aplicação. Um " pisca led ". Vamos elaborar uma rotina de atraso de aproximadamente 1 segundo. depois montar um circuito com o PIC para ver o LED apagar e acender em intervalos de 1 segundo. Lembre-se 1 segundo é um milhão de microsegundos, e que também pode ser dez vezes 100 milisegundos.

Passo 01

Criar a pasta para o projeto "piscaled" se não souber volte nas aulas anteriores.

faça o código fonte. para fazer o download clique aqui

c:_pic\piscaled\piscaled.asm

```
;######################   TITULO DO SEU CODIGO FONTE   #######################
;#                                                                          #
;#        Rotina de delay (atraso) de 1s utilizando apenas instrucoes       #
;#        usado para um circuito de pisca led                               #
;############################################################################

;----------------------DIRETIVAS AO COMPILADOR------------------------------

        LIST      P=16F84A
        RADIX     DEC
        INCLUDE   <P16F84A.INC>
        __CONFIG  _XT_OSC & _CP_OFF & _WDT_OFF & _PWRTE_ON


;------------------VARIAVEIS E EQUIVALENCIAS---------------------------------


x               equ     0ch     ;variavel auxiliar no delay
milisegundo     equ     0dh     ;variavel auxiliar no delay
seg             equ     0eh     ;variavel auxiliar no delay

;-------------------INICIALIZAÇÃO DO PROGRAMA--------------------------------

        clrf    portb           ; Faz todos os bits de portb = zer0
        bsf     status,rp0      ; vai para o banco 1 da RAM
        clrf    trisb           ; Faz portb todo saida
        bcf     status,rp0      ; volta ao banco 0 da RAM

;-------------------PROGRAMA PRINCIPAL---------------------------------------

repete
        bsf     portb,7         ; Faz bit 7 do portb=1 (apaga o LED)
        call    s1              ; Chama a subrotina s1 (nosso delay de 1s)
        bcf     portb,7         ; Faz bit 7 do portb=0 ( acende o Led)
        call    s1              ; Chama a subrotina s1 (nosso delay de 1s)
        goto    repete          ; Vai para o label repete
;------------------------------- SUB ROTINAS --------------------------------

;----- inicio da rotina de 1 segundo

s1
        movlw   10
        movwf   seg             ; Até aqui incluindo o call gastaram-se 4 us.

ms100
        movlw   100             ; 1 us Carrega milisegundo com 100
        movwf   milisegundo     ; 1 us

ms1
        movlw   249             ; carrega x com o valor 249 (decimal)
        movwf   x               ; 2 us ( 1 do movlw e 1 do movwf )
ms2
        nop                     ; + 1 us
        decfsz  x               ; + 1 us (no último eh 2 e pula p/ decfsz seg)
        goto    ms2             ; + 2 us, total 4us.(no último não passa aqui)
                                ; (4 us x 249)-1  totaliza 995 us
        decfsz  milisegundo     ; +1 us (na última pasagem 2 us)
        goto    ms1             ; +2 us (na última passagem pula)
                                ; total ms1 (1000 x 100)-1 = 99999
        decfsz  seg             ; + 1 us (no último eh 2 e pula p/ return)
        goto    ms100           ; +2 us (na última passagem pula)

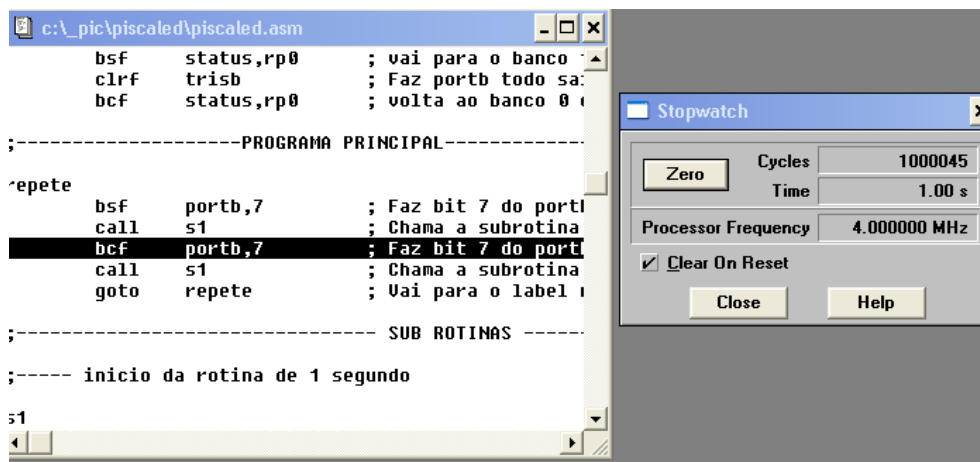
                                ; Total antes do return: 4us (inicio) +10 X 100004
                                ; {(1 do movlw100 + 1 do movwf milisegundo + 99999
                                ; da rotina ms1 + 1 do decfsz seg + 2 do goto)}- 1
                                ; da última passagem Total = 1000043 us
        return                  ; + 2 us retorna da sub-rotina apos 1.000.045 us
                                ; aproximadamente 1 segundo
;-----fim da rotina de 1 segundo

;---------------------------- fim do programa --------------------------------
        end
```

Passo 02

Faça o novo projeto conforme já foi ensinado, compile, faça a simulação para conferir o tempo. Coloque a janela de Stopwatch, vá simulando passo a passo com F7, depois de ter resetado com F6, faça a tarja chegar no primeiro Call, coloque um break point na linha de baixo no bcf portb,7, zere o stopwatch nesse ponto. Coloque para simular com o F9 ou o ícone do semáforo verde.

Esse vai demorar um pouco, pois tem que simular um pouco mais de um milhão de ciclos de máquina. quando terminar tem que ficar como a tela abaixo:



Como exercício, tente explicar como chegou-se a esse número de ciclos de máquina. se não conseguir tem o comentário no código fonte, que vai ajudar.

Passo 03

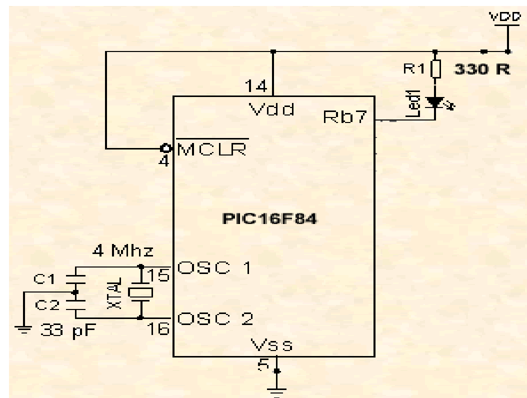
Vamos gravar o pic com o nosso programa. Pegue o gravador de PIC que já foi montado na aula anterior, ligue na fonte, ligue o cabo de comunicação serial ao computador, espete o PIC no lugar correto e vamos começar.

Você vai precisar de: PIC16f84 um Proto board, uma fonte de 5V, 1 cristal de 4 MHz, dois capacitores de 33pf, um resistor de 330 ohm e um led

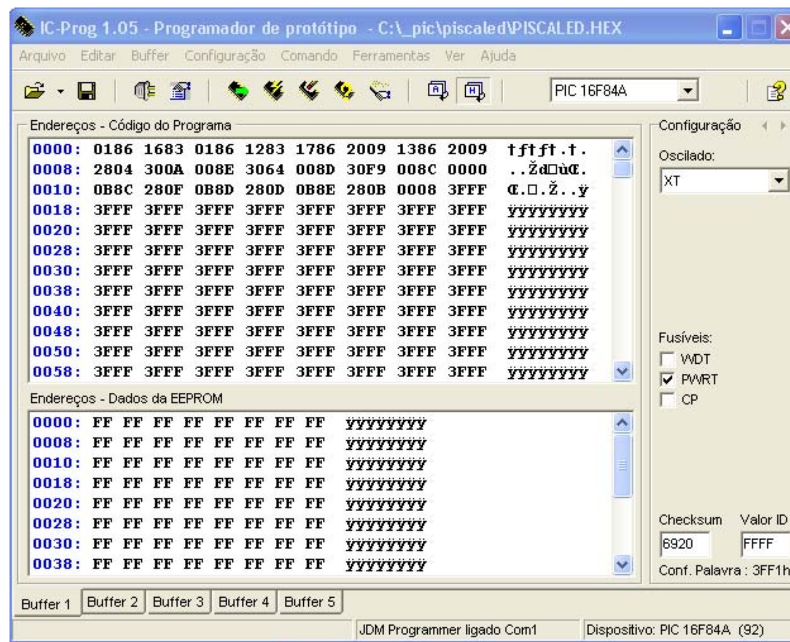


Passo 04

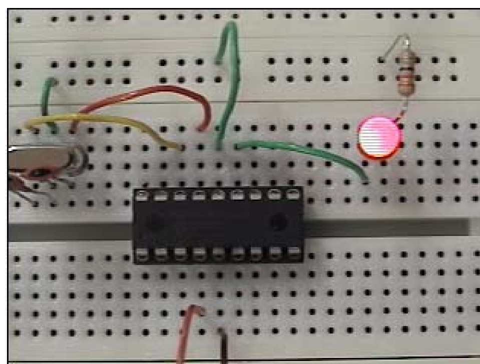
Vamos montar o circuito que está no esquema abaixo:



Abra o programa ICPROG, selecione o PIC16F84A, abra o arquivo piscaled.hex que o MPlab gerou e clique em gravar o dispositivo(ícone do raio)



Tire o pic do gravador e recoloque no circuito, ligue a alimentação... o LED vai começar a piscar em intervalos de 1 segundo.



Exercício

Com o mesmo circuito montado grave o PIC com programa delayt0.hex, verifique o que acontece. O led deverá piscar mais rápido.

Invente outros tempos e tente fazer as rotinas.

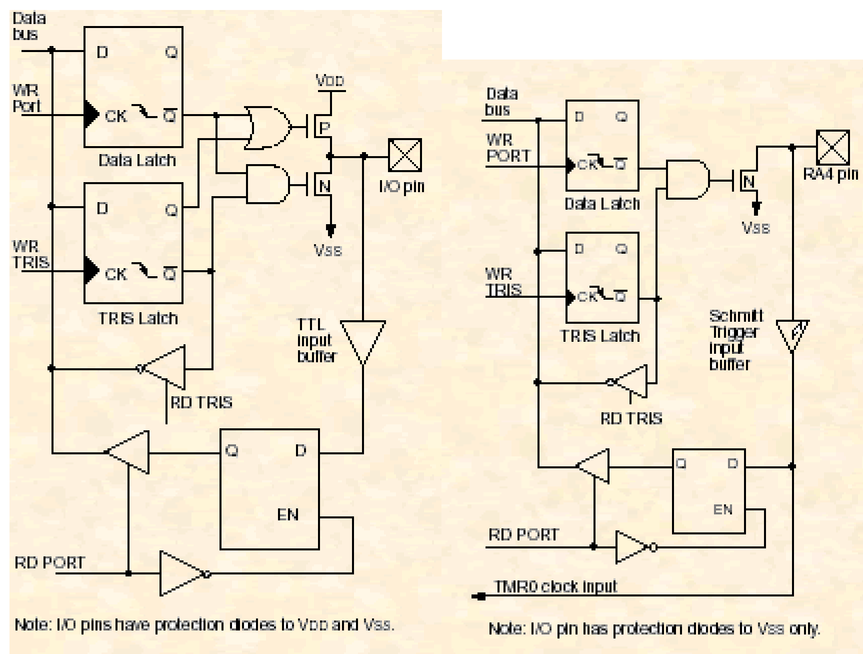
Aplicação de entradas e saídas digitais

O PIC16F84 possui 13 pinos que podem ser configurados como entrada ou saida digital. Primeiramente vamos lembrar da aula 10, sobre o hardware do PIC. Vamos rever como é o circuito dos pinos:

PORTA

O PORTA pode fornecer no máximo 50 mA e drenar no máximo 80 mA. Se você exigir o máximo de cada pino vai ultrapassar o valor limite do port, portanto tome cuidado ao calcular a demanda de corrente dos seus circuitos. Outro detalhe importante sobre o consumo de corrente, é o máximo de corrente permitido nos pinos VDD e VSS, no VDD o máximo é 100 mA e no VSS é 150 mA.

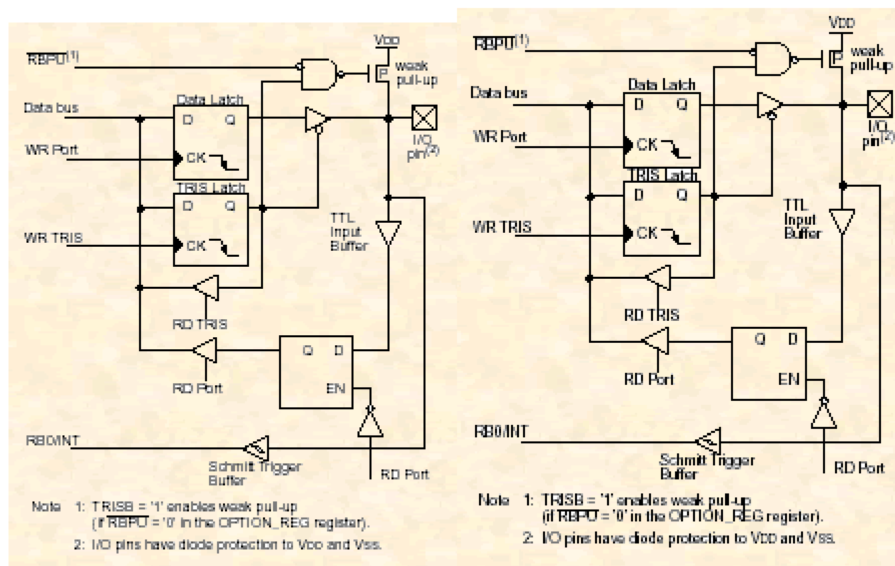
RA0 - RA3..RA4



PORTB

O PORTB pode fornecer no máximo 100 mA e drenar no máximo 150 mA. Se você exigir o máximo de cada pino vai ultrapassar o valor limite do port, portanto tome cuidado ao calcular a demanda de corrente dos seus circuitos.

RB0 - RB3...RB4 - RB7



Nome do pino	Port e bit	Tipo de entrada	Tipo de saída	Função
RA0	PORTA,0	TTL	Drive full CMOS	Entrada/Saída
RA1	PORTA,1	TTL	Drive full CMOS	Entrada/Saída
RA2	PORTA,2	TTL	Drive full CMOS	Entrada/Saída
RA3	PORTA,3	TTL	Drive full CMOS	Entrada/Saída
RA4	PORTA,4	ST	Mosfet dreno aberto	Entrada/Saída ou entrada de clock externo para o Timer0
RB0	PORTB,0	TTL / ST (1)	Buffer Tristate	Entrada/Saída (Resistor de pull up programável) ou entrada da interrupção externa
RB1	PORTB,1	TTL	Buffer Tristate	Entrada/Saída (Resistor de pull up programável)
RB2	PORTB,2	TTL	Buffer Tristate	Entrada/Saída (Resistor de pull up programável)
RB3	PORTB,3	TTL	Buffer Tristate	Entrada/Saída (Resistor de pull up programável)
RB4	PORTB,4	TTL	Buffer Tristate	Entrada/Saída (Resistor de pull up programável)
RB5	PORTB,5	TTL	Buffer Tristate	Entrada/Saída (Resistor de pull up programável)
RB6	PORTB,6	TTL / ST (2)	Buffer Tristate	Entrada/Saída (Resistor de pull up programável) . Pino usado durante a programação "clock"
RB7	PORTB,7	TTL / ST (2)	Buffer Tristate	Entrada/Saída (Resistor de pull up programável).Pino usado durante a programação "dados"

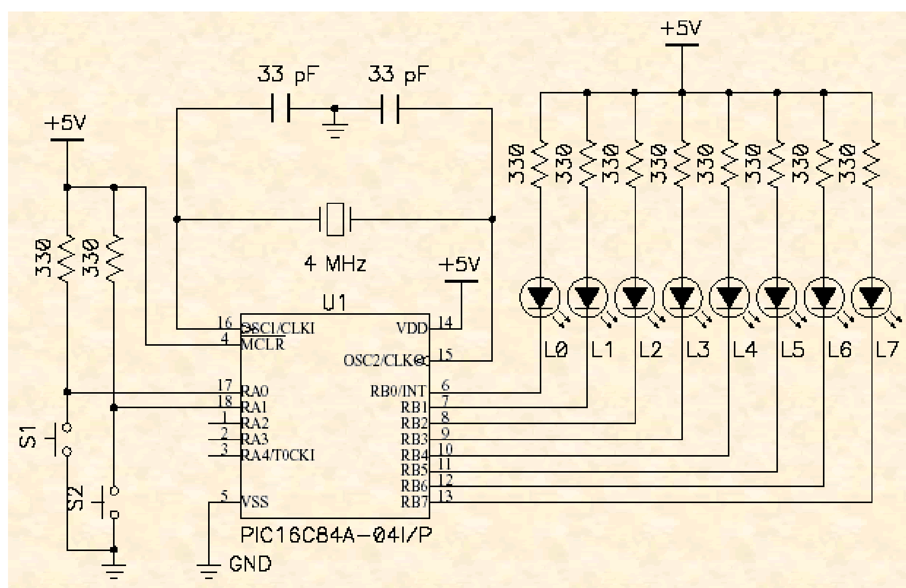
Legenda: TTL = entrada TTL, ST = entrada com Schmitt Trigger

Notas (1): A entrada é schmitt trigger quando programado a interrupção externa (2): A entrada é schmitt trigger durante a programação do PIC.

Características elétricas

Característica	TTL	ST	Full CMOS	Buffer Tristate	Mosfet dreno aberto
Entrada Nível Baixo	0,16 VDD	0,2 VDD	-	-	-
Entrada Nível Alto	0,25 VDD + 0,8V	0,8 VDD	-	-	-
Corrente típica de entrada	+/- 1 uA	+/- 1 uA	-	-	-
Tensão de saida Nível Baixo	-	-	0,6 V	0,6V	-
Tensão de saída Nível Alto	-	-	VDD - 0,7V	VDD - 0,7V	Máx 8,5 V
Máxima corrente de saída drenando	-	-	25 mA	25 mA	25 mA
Máxima corrente de saída Fornecendo	-	-	20 mA	20 mA	-
Corrente típica no Resistor de pull up interno no PORTB	-	-	-	250 uA	-

Nossa aplicação vai usar o PORTB como saída digital, e vai estar conectado a 8 led´s. Vamos usar o PORTA como entrada digital e vai estar conectado a duas chaves. Para a aplicação temos que montar o circuito abaixo:



Note o seguinte: preferi ligar os led´s com o catodo nos pinos para o PIC drenar a corrente, onde a capacidade é maior que fornecer, então com nível zero no pino o Led acende e com nível 1 o Led apaga. As chaves estão ligadoas nos pinos RA0 e RA1 usando um resistor de pull up externo para manter 5V quando a chave está solta, portanto, quando a chave está pressionada é nível zero que teremos nos pinos. O /MCLR foi ligado diretamente ao VCC pois não vamos usar o Reset.

A primeira proposta para este circuito, ou hardware é a seguinte:

- Condição inicial : Todos os led´s apagados.
- De acordo com as chaves os Leds se comportarão assim:

Entradas	Saídas

S1	S2	L0	L1	L2	L3	L4	L5	L6	L7
Solta	Solta	Apagado	Apagado	Apagado	Apagado	Apagado	Apagado	Apagado	Apagado
Solta	Pressionada	Aceso	Aceso	Aceso	Aceso	Apagado	Apagado	Apagado	Apagado
Pressionada	Solta	Apagado	Apagado	Apagado	Apagado	Aceso	Aceso	Aceso	Aceso
Pressionada	Pressionada	Aceso	Aceso	Aceso	Aceso	Aceso	Aceso	Aceso	Aceso

Passando essa tabela para níveis lógicos nos pinos do PIC

Entradas		Saídas							
RA0	RA1	RB0	RB1	RB2	RB3	RB4	RB5	RB6	RB7
1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1
0	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Com a proposta na mão, agora temos que pensar no software.

Para as saídas a gente já fez algumas aplicações anteriores, usando a instrução *BSF f* e o *BCF f*, que são instruções que alteram um único bit por vez. Quando trabalhamos com diversos bits podemos alterar o registro inteiro, ou o PORT inteiro, usando duas instruções:

- MOVLW k (Mover o literal k para o registrador W) podemos escrever k em binário, isso facilita quando queremos alterar os ports.
- MOVWF f (Mover o conteúdo de W para o registrador F). Como no PIC não podemos mover um registro diretamente para outro, ou um literal diretamente para um registro , temos que usar este recurso: primeiro coloca-se em W e depois de W coloca-se onde queremos.

  No nosso exemplo, quando S1 estiver solta e S2 estiver pressionada termos que ter o portb com 00001111 isso é bem fácil de fazer com software.

```
movlw B'0001111'

movwf portb
```

Primeiro colocamos o número literal binário 00001111 em w e depois em portb. Note a facilidade de se poder escrever diretamente o número binário para o compilador, a regra para se screver em binário é colocar o B na frente e o binário entre apóstrofos .

Para as entradas digitais, podemos fazer um teste bit por bit, ou seja, chave por chave usando as instruções *btfss f* ou *btfsc f* como vimos na aula 14

```
c:\_pic\chaves\chaves.asm
;#####################    TITULO DO SEU CODIGO FONTE    #######################
;#                                                                             #
;#        Rotina com entrada e saidas digitais  2 chaves e 8 leds              #
;#                                                                             #
;###############################################################################

;-----------------------DIRETIVAS AO COMPILADOR------------------------------

        LIST      P=16F84A
        RADIX     DEC
        INCLUDE   <P16F84A.INC>
        __CONFIG  _XT_OSC & _CP_OFF & _WDT_OFF & _PWRTE_ON


;-------------------VARIAVEIS E EQUIVALENCIAS---------------------------------

S1      equ     0               ;O compilador vai substituir S1 por 0
S2      equ     1               ;O compilador vai substituir S2 por 1

;--------------------INICIALIZAÇÃO DO PROGRAMA---------------------------------

        clrf    portb           ; Faz todos os bits de portb = zero
        bsf     status,rp0      ; vai para o banco 1 da RAM
        clrf    trisb           ; Faz portb todo saida
        bcf     status,rp0      ; volta ao banco 0 da RAM

;--------------------PROGRAMA PRINCIPAL---------------------------------------

testa_chaves
        btfss   porta,S1        ;testa se S1 está solta
        goto    chave1          ;s1= pressionada vai para a rotina da chave 1
        btfss   porta,s2        ;s1=solta testa se S2 esta solta
        goto    chave2          ;S2= pressionada vai para a rotina da chave 2
        movlw   b'11111111'     ;S2 = solta, então se chegou aqui é por que S1
                                ;S2 estão soltas Faz o w igual a binário 111111
        movwf   portb           ;Carrega o portb com w
        goto    testa_chaves    ;testa novamente as chaves
;-------------------------------- SUB ROTINAS ---------------------------------

chave1
        btfss   porta,s2        ;testa se S2 esta solta
        goto    chaves_juntas   ;S2 = pressionada. então se chegou aqui é porqu
                                ;S1 e S2 estão pressionadas. então vai para
                                ;chave_juntas
        movlw   b'00001111'     ;Carrega o w com o binário 00001111
        movwf   portb           ;Carrega o portb com w
        goto    testa_chaves    ;volta a testar as chaves

chave2
        movlw   b'11110000'     ;Carrega o w com o binário 11110000
        movwf   portb           ;Carrega o portb com w
        goto    testa_chaves    ;volta a testar as chaves

chaves_juntas
        movlw   b'00000000'     ;Carrega o w com o binário 00000000
        movwf   portb           ;Carrega o portb com w
        goto    testa_chaves    ;volta a testar as chaves


;----------------------------- fim do programa --------------------------------
        end
```

Crie um novo projeto chamado chaves. Compile o programa, coloque uma janela de watch windows para observar os registros porta e portb em binário. Vamos aprender a simular as entradas no MPlab.

O Mplab possui uma ferramenta muito boa para a simulação das entradas de sinal nos pinos. essa janela esta no menu Debug > Simulator stimulus > assynchronous stimulus. ela tem possibilidades de simular até 12 pinos e podemos usar os seguintes métodos:Pulse Low Hight e Toggle, prefiro usar o toggle que a cada apertada no botão ele troca de estado lógico o pino a ele atribuido. Veja a apresentação abaixo:

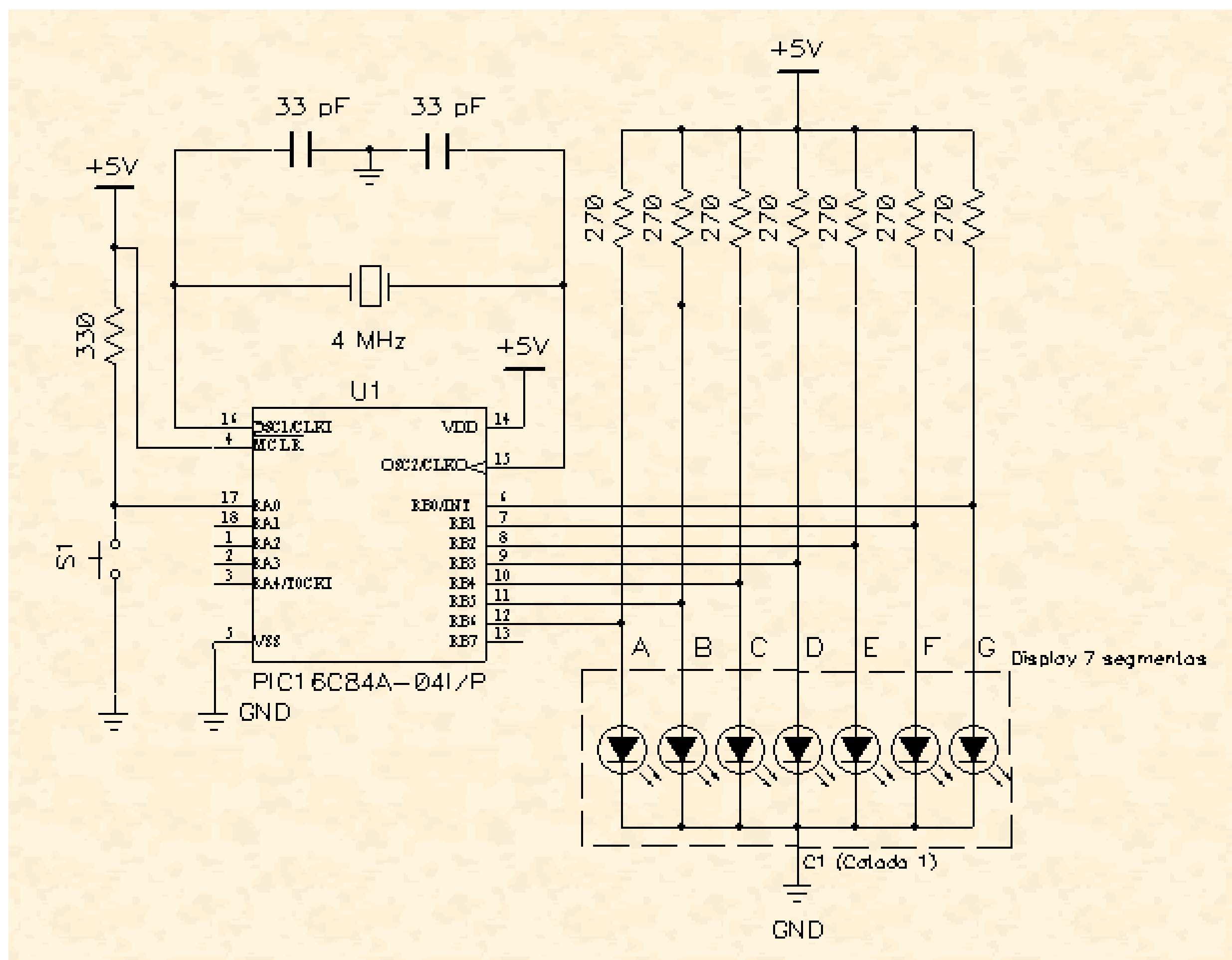
Controlando um display de 7 segmentos

O display de sete segmentos, nada mais é do que um invólucro com sete led´s com formato de segmento, posicionados de tal forma a possibilitar a formação de números decimais e algumas

letras utilizadas no código hexadecimal. Nesta aplicação vamos usar um display, que será ligado as saídas do PIC 16F84..

Abaixo temos os circuito com o PIC 16F84 que iremos montar para esta aplicação . O display usado, é o de catodo comum, portanto as saídas do PIC deverão ter nível lógico 1 quando quiser acender um segmento.

Obs. aqui tem uma "caca" o display teria que ser Anodo Comum, com o anodo ligado a Vcc e as resistências nas portas, para drenarem a corrente. eu errei no esquema elétrico. Para o catodo comum as portas podem alimentar direto os segmentos...(desculpem a falha...qdo sobrar um tempinho eu faço a correção)



Vamos definir quais os valores que o display deve mostrar, montando uma tabela com os segmentos que devem acender conforme o número que se deseja mostrar, abaixo temos a disposição física dos segmentos do display e a tabela.Os números que vão aparecer são de 0 a 9, acompanhe como fica a tabela.

Valor Desejado	Sa	Sb	Sc	Sd	Se	Sf	Sg
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

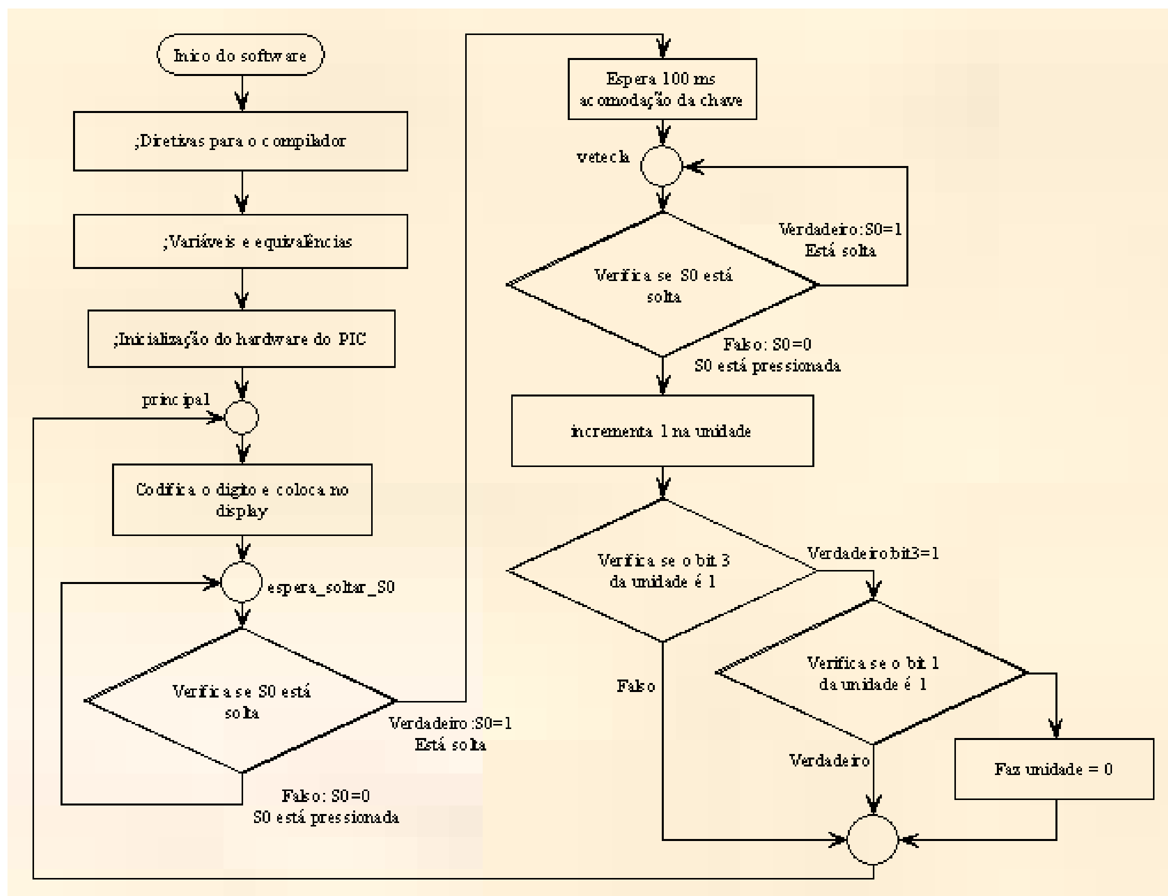
A aplicação que iremos fazer, consiste no seguinte: Temos uma chave S1 ligada em RA0 do PIC. Quando ligamos o circuito o display mostra o digito Zero, A cada toque em S1, após soltar a chave, adiciona-se uma unidade no display, e quando chegar a dez ele volta para Zero. Faça um novo projeto usando o codigo fonte disp7_01.asm

A Tabela que codifica o digito em sete segmentos:

Esse Programa usa a instrução RETLW, com essa instrução é possível criar tabelas no software, essa instrução sozinha retorna para onde foi chamada com o W igual ao valor literal k que está na sua frente. O segredo é endereçar o Contador de Programa para a posição desejada na tabela. Isso a gente consegue adicionando ao PCL, registrador que mantém o endereço da memória de programa da próxima instrução a ser executada, sabemos que o endereço da instrução não cabe em um byte, o contador de programa é armazenado em 2 registradores o PCLATH, parte alta do endereço e o PCL, parte baixa do endereço, Então se sua tabela for maior que 255 posições tem que usar os dois registros. Na maioria das aplicações, as tabelas são menores que 255, então é só somar a posição da tabela que você quer o retorno ao PCL. No nosso caso a tabela tem 10 posições. digitos de 0 a 9. No software a rotina chamada "codifica", tem como a primeira instrução a adição da posição da tabela ao PCL, e as próximas posições são os retlw com o valor que queremos retornar.

Outra situação interessante no software, é o teste para ver se o digito chegou a dez.

Veja o Fluxoframa do software.



O código fonte está comentado, Utilizando o fluxograma, as ferramentas de simulação do Mplab, estude atentamente cada passo do programa e o que você não entender, me pergunte.

Obs.: Para simular lembre-se de colocar no início do programa a diretiva

#define simula

E na rotina " delay de 100 milisengundos"

ms100:

#ifdef simula

return

#endif

movlw 100

movwf tempo

Uso das Interrupções e Sleep

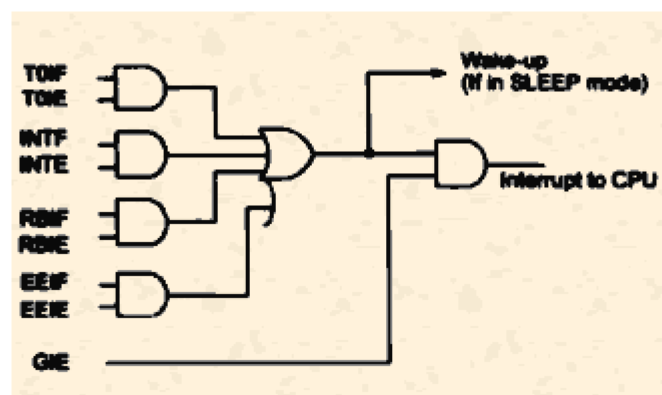
O PIC 16F84 possui quatro sinalizadores de interrupção, sendo três presentes no registro INTCON:

INTIF - Interrupção externa
T0IF - Overflow do Timer 0
EEIF - Fim de escrita na EEPROM, que se encontra no registro EECON1
RBIF - Mudanças nos pinos RB4 a RB7

Temos o GIE (Global Interrupt Enable - habilita interrupção global), que controla todas as interrupções, ou seja, ela é uma chave geral. Se GIE estiver em 0, nenhuma interrupção será atendida.

Mas se GIE estiver em 1, as interrupções que estiverem habilitadas (RBIE, INTE,...) serão atendidas pela CPU.

Circuito lógico de atendimento de interrupção:



Atendimento das Interrupções

Ao aceitar um pedido de interrupção, a CPU faz GIE = 0, para inibir as outras interrupções de poderem ocorrer durante a execução de uma em execução, salva no Stack o endereço de retorno (a próxima após o PC atual, isto é, PC +1), e depois desvia para o endereço 0004h da memória de programa.

Ao encontrar uma instrução RETFIE (Return From Interrupt - volta da interrupção) a CPU finaliza a interrupção, recuperando o PC do stack e novamente fazendo GIE = 1 para um próximo pedido de interrupção.

A instrução RETFIE é somente usada para retorno de interrupção.

Para as interrupções externas, o atraso no início da rotina de interrupção é de 3 a 4 ciclos de instrução.

Fontes de Interrupção

Todo pedido de interrupção desvia para o endereço 004h, por isso, o software deve verificar nos bits sinalizadores de interrupção, qual está setado (1), determinando a fonte de interrupção. Estes bits sinalizadores são os bits RBIF, INTF, T0IF contidos no registro INTCON e o EEIF, contido no registro EECON1, como já foi visto.
Os bits sinalizadores, mais comumente chamados de FLAGs, sempre serão ativados pela respectiva fonte de interrupção, mesmo o GIE estando desabilitado, ou seja, em 0.

Com base no parágrafo anterior, antes de habilitar uma interrupção, o usuário deve certificar se a mesma já não está sendo requisitada, pois seu atendimento pode ser indesejável.

Interrupção Externa:

A interrupção externa, pino RB0/INT, é sempre ativada por borda: podendo ser por borda de subida, colocando nível lógico 1 no bit INTDG (bit 6 do OPTION_REG) ou por borda de descida colocando nível lógico 0 no bit INTDG.

Quando o tipo de borda escolhida aparecer no pino RB0/INT, o bit INTF (bit 1 do registro INTCON) será levado ao nível lógico 1. Esta interrupção é sempre habilitada pelo bit de controle INTE (bit 4 do INTCON), se 0 está desabilitada, se 1 está habilitada. O bit de flag INTF deve ser zerado por software antes de retornar da interrupção.

A interrupção externa pode tirar o processador do modo SLEEP, somente se o bit INTE estiver habilitado (em 1) antes do processador entrar no modo SLEEP. O estado do bit GIE definirá como o processador se desviará para o vetor de interrupção, seguido de um Wake-up (acordar, sair do modo SLEEP)

Interrupção pelo Timer 0

Um estouro no TMR0 (FF 00h ou 255d 000d) irá levar a nível lógico 1 o bit T0IF (bit 2 do INTCON). Se a interrupção no bit T0IE (bit 5 do INTCON) estiver habilitada (1), o processador desviará para o vetor de interrupção. Caso contrário (0), somente ocorrerá a sinalização. O bit T0IF deve ser zerado por software

Interrupção no PORTB

Uma mudança na entrada em um dos pinos RB4 a RB7 do PORTB , levará a nível lógico 1 o bit RBIF (bit 0 do INTCON). Se o bit de habilitação de interrupção RBIE (bit 3 do INTCON) estiver em 1, desviará o processador para o vetor de interrupção. O bit de flag RBIF deve ser zerado por software.

Nota: Para que a mudança de estado, nos pinos de I/O, seja reconhecida, a largura do pulso deve ser no mínimo o tamanho de um ciclo de máquina (Tcy).

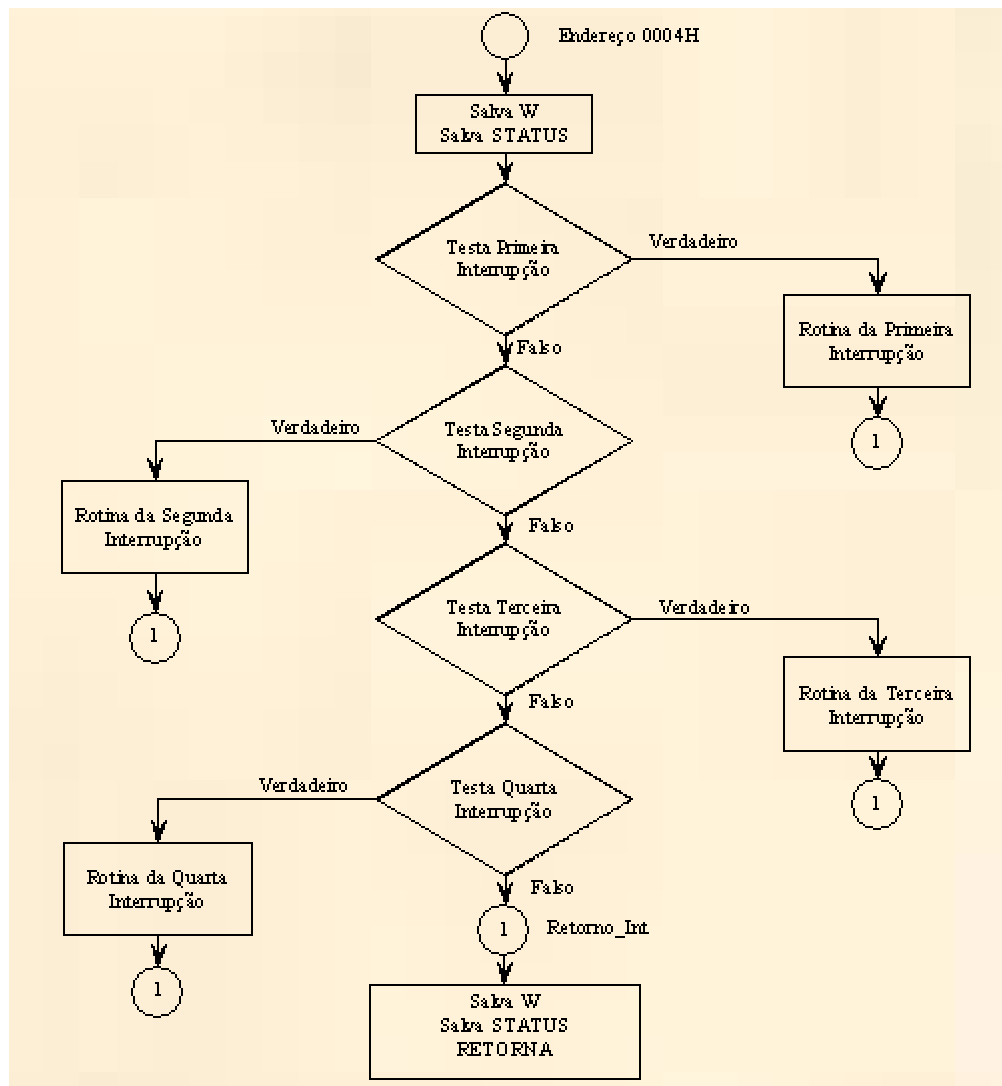
Interrupção pela Escrita na EEPROM

Ao completar o ciclo de escrita de dados na EEPROM, o bit de flag EEIF (bit 4 do EECON1) será levado ao nível lógico 1. Se o bit de interrupção EEIF (bit 6 do INTCON) estiver habilitado (1), a CPU desviará para o vetor de interrupção.

Roteiro Básico para atender as Interrupções:

1. Primeiramente a rotina de atendimento tem que estar no endereço 0004H da memória de programa. As interrupções a serem atendidas devem estar habilitadas,registro INTCOM, bem como a "chave geral" das interrupções, GIE .
2. Se o processador chegou no endereço 0004H é por que houve uma interrupção, não sabemos qual ainda. Durante uma interrupção, somente o valor do contador de programa PC é salvo no Stack. Devemos salvar os valores de registros que podem ser afetados durante as rotinas de interrupções, como bits de flag Z, C, RP0 e outros , presentes no STATUS, além do registro W. Para isto temos abaixo como exemplo, algumas linhas importantes e necessárias para serem colocadas no início da interrupção, salvando W e STATUS.
    - endereço 0004h
    - movwf w_temp ; salva conteúdo do registro W em w_temp (endereço temporário)
    - movf status,w ; coloca o status em W
    - movwf status_temp ; salva o conteúdo do registro W em atatus_temp (endereço temporário)
3. Depois de salvo o W e o STATUS vamos testar qual a interrupção que foi acionada, para isso testamos todos os flags das interrupções habilitadas, nesse ponto é que definimos qual a escala de prioridades das interrupções, não raro, duas ou mais interrupções podem ser acionadas ao mesmo tempo, a de maior prioridade, é a que testamos em primeiro lugar e as outras na sequência de prioridade.
4. Criamos uma rotina para cada interrupção
5. Criamos a rotina a rotina para retornar da interrupção, recuperando o W e o STATUS.

Veja o código fonte abaixo, serve como um modelo (ABRIR PROGRAMA INT_.asm)



SLEEP

Power-down Modo SLEEP

O sistema pode ser colocado em baixo consumo (SLEEP), e mais tarde voltar a operação normal (Wake-up).

SLEEP

O modo Power-down (tradução - desligamento, mas na verdade entra em baixo consumo Standby) é acionado pela execução da instrução SLEEP. Se habilitado, o Watchdog timer é zerado (mas mantém rodando), O bit (bit 3 do STATUS) é zerado, o (bit 4 do STATUS) é levado a nível lógico 1, e o driver oscilador é desligado. As portas de I/O são mantidas como estavam, antes da instrução SLEEP ser executada (Alto, baixo ou alta impedância).

Esta característica aliada ao Pull-ups do PORTB e a interrupção nas mudanças de estado, permitem o projeto de sistemas para funcionar a bateria de longa duração, como controles remotos em geral.

Saindo (Wake-up) do modo SLEEP

O sistema pode sair do modo SLEEP através dos seguintes eventos:

- Reset externo pelo pino
- WDT Wakw-up (se o WDT estiver habilitado)

- Interrupção pelo pino RB0/INT, por mudanças de estado no PORTB (RB4 a RB7), ou pelo sinal de escrita completa na EEPROM.

O primeiro evento citado (reset) irá causar um reset no sistema.

Os outros dois eventos são considerados uma continuação de um programa em execução. Os bits e são usados para determinar quem causou o reset dentre os dois eventos. O bit , o qual é levado a nível lógico 1 no Power-up, é levado a nível lógico 0 quando o SLEEP é ativado. O bit é levado ao nível lógico 0 se um WDT time-out ocorrer (causando o Wake-up). Enquanto a instrução SLEEP está sendo executada, a próxima instrução após o SLEEP está pré-determinada (PC+1). Para o sistema sair do Sleep através da interrupção, esta deverá estar habilitada (1). O Wake-up ocorre sem levar em consideração o estado do bit GIE, portanto, na interrupção, o sistema se comportará de duas formas conforme o GIE:

- se GIE = 0 - ao ocorrer um wake-up, o sistema vai executar a instrução imediata (PC+1) a instrução SLEEP, continuando a execução normal do programa.
- Se GIE = 1 - Ao ocorrer o wake-up, o sistema vai executar a instrução imediata (PC+1) a instrução SLEEP e depois irá desviar para o endereço de vetor da interrupção (0004h). Para que uma interrupção seja atendida logo após o Wake-up, recomenda-se colocar uma instrução NOP após a instrução SLEEP.
  Saindo da interrupção (Wake-up) através da interrupção

Wake-up através da interrupção

Se uma interrupção ocorrer antes de uma instrução Sleep, a instrução Sleep se comportará como um NOP, o WDT não será zerado, o não será setado e o não será levado a nível lógico zero, ou seja, como se a instrução SLEEP nunca tivesse existido.
Se a interrupção ocorrer logo após a instrução SLEEP, esta será executada (modo SLEEP executado) e em seguida sairá do modo SLEEP, para atender a interrupção, zerando o WDT, = 1 e = 0, assim o SLEEP se comportou como uma simples instrução.
Para saber se realmente o SLEEP ocorreu, basta verificar se o bit está com nível lógico 0, se não estiver, o SLEEP se comportou como um NOP.
Embora o watchdog seja zerado pelo SLEEP, devemos por segurança, incluir a instrução CLRWDT
imediatamente antes da instrução SLEEP, pois se por coincidência o watchdog pode estar gerando um Timer-out durante a execução do SLEEP.

Vamos fazer uma aplicação bem simples, Uma Chave S1 ligada na entrada de interrupção, um Led ligado em RB7, em zero acende o Led. Ao ligar o circuito o Led acende por 1 segundo, aí o Pic entra em modo Sleep. Quando apertar S1, que é a interrupção externa o led começa a piscar, solta S1 volta ao sleep

Faça um novo projeto com o fonte int_01.asm Clique aqui para download, arrume as janelas como na apresentação abaixo, e estude todos os passos do programa. é uma aplicação simples, mas mostra todos os passos para se usar as interrupções.

(Errata:: o Thiago, achou um erro no codigo fonte, já concertei no código fonte, mas dá muito trabalho refazer a animação.. então por favor desconsiderem o erro na animação...(não compromete nenhum conceito.) esse erro é o do preguiçoso, que em vez de fazer as rotinas novamente, fica fazendo Ctrl+C - Ctrl+V e remendando... na rotina de 1 segundo eu troquei y por x.)

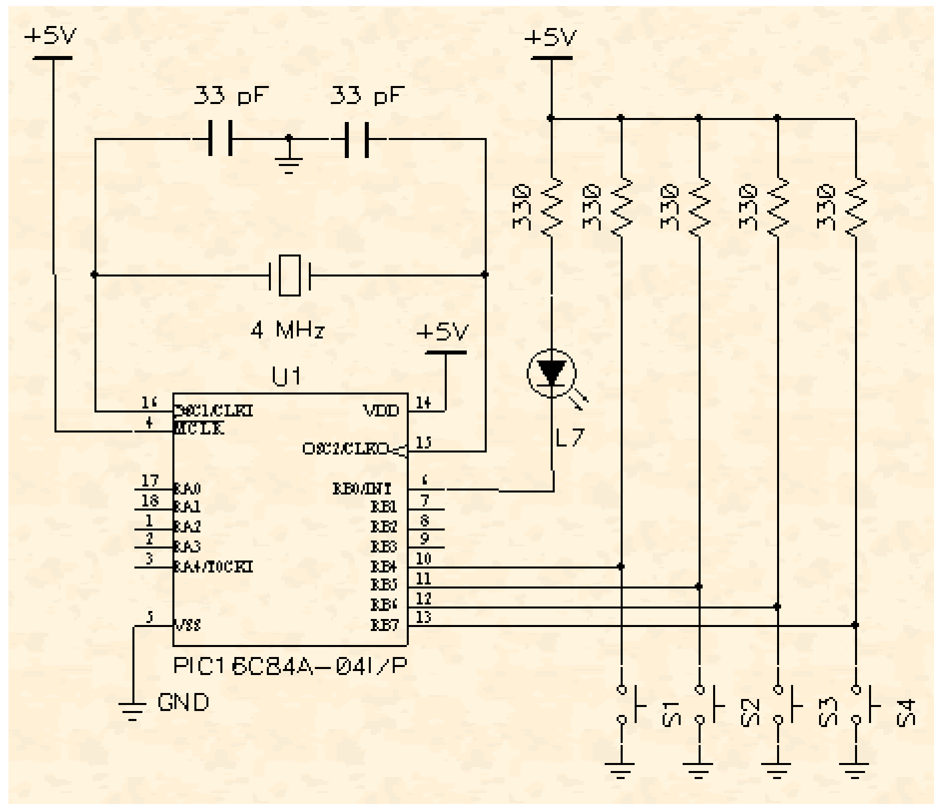
Interrupção de RB0

A teoria sobre as interrupções, é a mesma da aula 18. Essa interrupção de RB0 está relacionada aos pinos RB4 a RB7, esses quatro pinos, quando habilitada a interrupção, ao mudarem de estado lógico, tanto faz se de 1 pra 0 ou de 0 pra 1, provocam um pedido de interrupção. Esta facilidade é importante em teclados, principalmente em controles remotos que trabalham com baterias, e no estado normal estão em sleep para economia de energia.

Basta uma tecla ser acionada e o PIC sai do sleep e atende a interrupção.

Vamos usar o mesmo modelo de Programa fonte da aula 19. O circuito dessa aplicação consiste em quatro chaves, S1, S2, S3 e S4 ligados em RB4, RB5, RB6 e RB7 respectivamente e um LED em RB0.

Quando ligamos o circuito o LED fica aceso por 1 segundo e depois o pic entra em standby, sleep, basta uma chave ser alterada e vai pra interrupção.



Fazer um novo projeto usando o fonte int_02.asm

Estudando o WatchDog

Temporizador WatchDog (WDT)

O temporizador Watchdog é livremente incrementado por um oscilador RC interno, que não requer nenhum componente externo. Este oscilador RC é separado do dispositivo oscilador presente no pino OSC1/CLKIN. Esse é meio pelo qual o WDT incrementa, mesmo se o dispositivo oscilador presente nos pinos OSC1/CLKIN e OSC2/CLKOUT estiver parado, como por exemplo, a execução de uma instrução SLEEP.

Em uma operação normal, um WDT time-out gerará um RESET no dispositivo. Se o dispositivo estiver em modo SLEEP, um WDT wake-up irá tirar o dispositivo do modo SLEEP e voltar a operação normal. O WDT pode ser permanentemente desabilitado, programando o bit de configuração WDTE, presente nos fusíveis de configuração, já comentado

O Watch Dog é utilizado para ficar vigiando o programa principal, se po algum motivo ele "travar" e não passar pela instrução que zera o timer wdt, o mesmo provoca um reset no Pic.

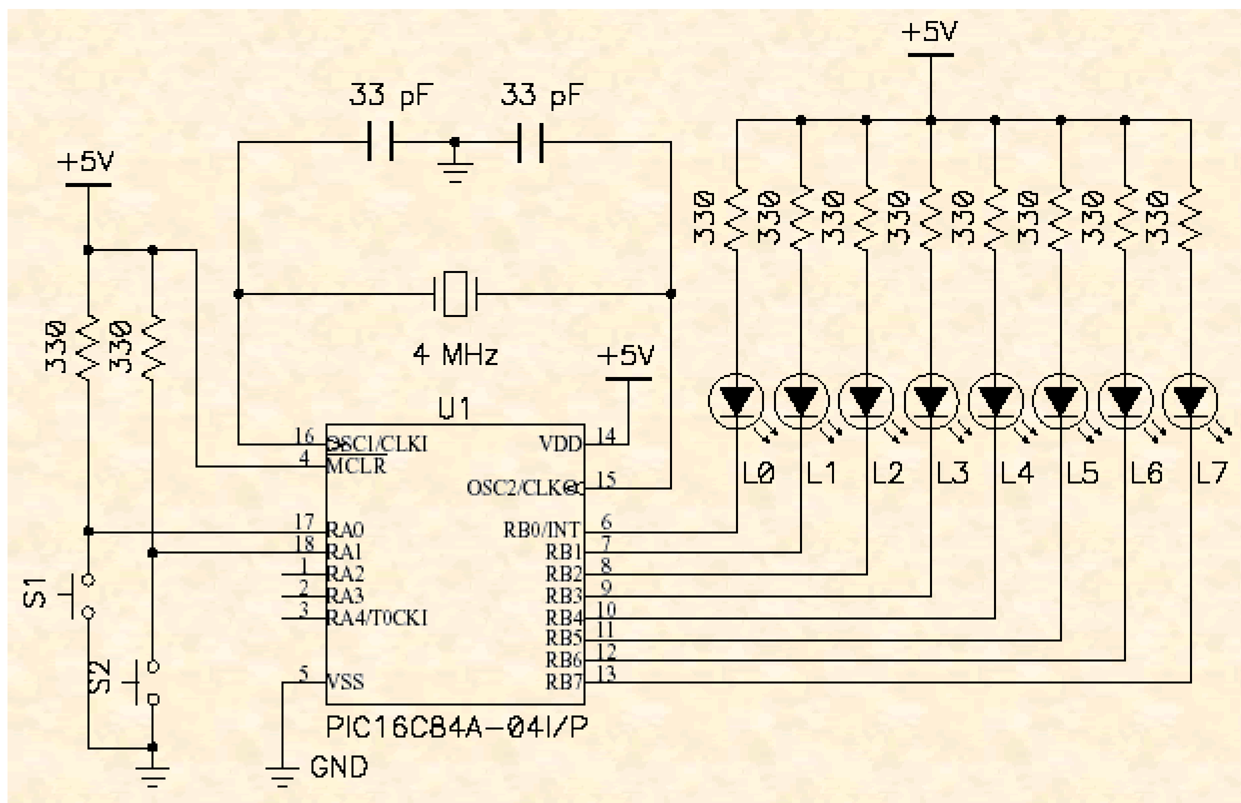
Período do WDT

O WDT tem um time-out (intervalo) de 18 ms, (sem o prescaler). O time-out pode variar com a temperatura, VDD e de componente para componente (Especificados em manual). Se desejar um Time-out maior, pode usar o prescaler, onde é possível conseguir uma taxa de 1:128, configurado através de programação do registro OPTION_REG (os bits PSA, PS2, PS1 e PS0). Assim, o time-out pode chegar a 2,3 segundos.

As instruções CLRWDT e SLEEP apagam o WDT e o postscaler (se WDT estiver habilitado) e previne de um WDT time-out, gerando uma condição de RESET.
Deve-se levar em consideração, na programação do WDT, as piores condições de trabalho, ou seja, VDD = ao mínimo, temperatura máxima e WDT prescaler), pois pode tomar alguns severos segundos antes do WDT Time-out ocorrer.

EXPERIMENTO

Para estudarmos o wdt vamos montar um circuito com 8 leds no PORTB e uma chave em RA0. O circuito ao ligar, ou no reset do pic, faz acender todos os leds durante 300 ms, logo após inicia-se uma sequência de acendimento de leds com tempo de 100 ms, a cada intervalo de acendimento o software espera o CLEAR WDT, que em nosso caso vamos fazer externamente através da chave. Então se ficarmos mais de 2,3 segundos sem pulsar a chave o WDT vai resetar o PIC. Nós poderemos observar isso vendo os led´s. Usaremos o mesmo circuito da aula 16, a única diferença é que não usaremos a chave S2.



Faça um novo projeto com o fonte wdt_01.asm, clique aqui para o download. Esse programa fica difícil a simulação devido aos grandes tempos envolvidos, a melhor forma é observar o circuito. Quando usamos o wdt, na linha do __config temos que colocá-lo em on, _wdt_on. No registrador OPTIONREG apontamos o prescaler para o wdt. Estude o código fonte passo a passo.

Estudando a EEPROM

Memória EEPROM de Dados

A memória EEPROM de dados pode ser lida e escrita durante a operação normal (com a tensão normal de alimentação). Esta memória não é diretamente mapeada no banco de registros, devendo ser endereçada através dos registros de funções especiais, sendo necessário quatro FSR para leitura e escrita em EEPROM. São eles: EECON1, EECON2 (registro não está implementado fisicamente), EEDATA e EEADR.

No EEDATA, armazenam os 8 bits (byte) para leitura ou escrita. No EEADR armazena o endereço da EEPROM que será acessado. No PIC 16F84 a EEPROM de dados tem um tamanho de 64 bytes, e seu endereço vai de 00h a 3Fh.

A escrita na EEPROM automaticamente grava sobre o dado armazenado anteriormente. A vantagem desta memória é que ao ser gravado uma informação, ela não se perderá ao desligar o sistema. Para uma escrita na EEPROM, gasta-se aproximadamente 10 mS, fator que "atrasa" o

sistema, mas para leitura gasta o mesmo que uma leitura na RAM (se o clock for 4 MHz gastará 1 S).

Registro EECON1

O registro EECON1 é para controle das operações com a EEPROM. Seu endereço é 88h. Valor após o reset: UUU0X00X. O U será lido como 0.

Propriedade	U	U	U	R/W	R/W	R/W	R/S	R/S
Reset	-	-	-	0	X	0	0	0
Bit	7	6	5	4	3	2	1	0
Nome	-	-	-	EEIF	WRERR	WREN	WR	RD

EEIF - Bit sinalizador de interrupção de fim de escrita
1 - Já acabou a escrita (zerado por software)
0 - não acabou de escrever

WRERR - bit sinalizador de erro ao escrever na EEPROM
1 - Escrita prematuramente interrompida (por reset ou Watch Dog)
0 - Operação de escrita completada

WREN - Bit de habilitação de escrita na EEPROM
1 - Permite o ciclo de escrita
0 - Inibe a escrita de dados na EEPROM

WR - Bit de controle de escrita
1 - Inicia o ciclo de escrita. Será zerado por hardware assim que a escrita for completada
0 - A escrita na EEPROM foi completada

RD - Bit de controle de Leitura na EEPROM
1 - Inicia uma leitura na EEPROM. (é zerado por hardware. Gasta um ciclo)
0 - Não inicia leitura na EEPROM

Os bits RD e WR podem ser lidos, mas por software só pode setar.

Para iniciar uma operação de leitura ou escrita, basta colocar os valores em EEADR e EEDATA (na escrita) e setar os bits RD ou WR conforme a operação desejada.

Em muitas aplicações há a necessidade de armazenar dados numa memória que não se apague quando desligamos a energia, (último numero discado de um telefone, canais de rádio, senhas de acesso etc.) o PIC 16F84 possui 64 bytes de EEPROM para dados que não perdem a informação com a falta de alimentação, mas para escrever nessa memória o PIC gasta um tempo grande: pode passar dos 10 ms, isto é 10.000 maior que a RAM com um cristal de 4 MHz

Estes 64 bytes não são diretamente endereçados como a RAM, mas sim através de 4 registros especialmente para elas que são:

- EEADR = endereço desejado para leitura ous escrita;
  de 00H a 3FH no F84 (64 posições)
- EEDATA= dado a escrever ou dado lido neste registro teremos que colocar o dado a ser gravado, ou simplesmente ler após o comando de leitura.
- EECON1= registro de controle 1 Possui 5 bits de controle ver registros especiais.
- EECON2= registro de controle 2 Esse registro não necessita controle por parte do usuário, é usado internamente durante a gravação.

Exemplo:
LENDO UM DADO: (endereço 10H da EEPROM, salvando no 14H da RAM)

Até aqui estava no banco 0 (RP1=0 sempre para o 16F84)

```
Bcf STATUS,RP0 ;banco 0 onde está o EEADR
Movlw 0x10 ;W=10H
Movwf EEADR ;EEADR=10H (endereço da eeprom)
Bsf STATUS,RP0 ;banco 1 onde está o EECON1
Bsf EECON1,RD ;inicia a leitura, leva um ciclo de máquina
Bcf STATUS,RP0 ;volto ao banco 0 onde está o EEDATA
Movf EEDATA,W ;W=EEDATA( w= 10H eeprom )
Movwf 0x14 ;14H da RAM = 10H EEPROM
```

ESCREVENDO UM DADO: (endereço 10H da EEPROM, dado=87H)

Até aqui estava no banco 0 (RP1=0 sempre para o 16F84)

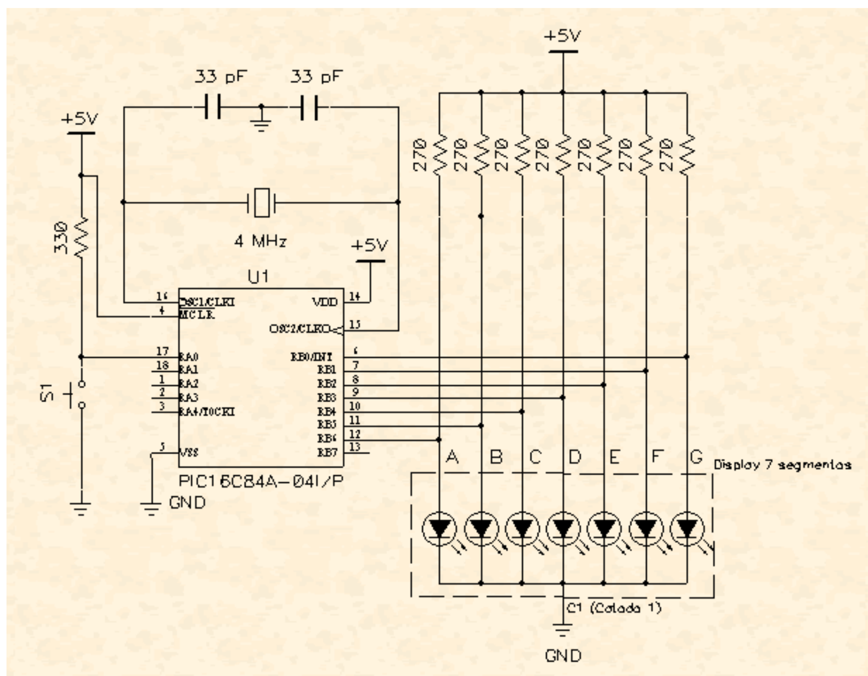
```
Bcf STATUS,RP0 ;banco 0 onde está o EEADR
Movlw 0x10 ;W=10H
Movwf EEADR ;EEADR=10H (endereço da eeprom)
Movlw 0x87 ;W=87H
Movwf EEDATA ;EEDATA=87H (dado a ser gravado)
Bsf STATUS,RP0 ;banco 1 onde está o EECON1
Bcf INTCON,GIE ;desabilita a interrupção para não haver erro durante a gravação
bsf EECON1,WREN ;Habilita a escrita na EEPROM
* movlw 0x55 ;W=55H
* movwf EECON2 ;EECON2 = 55H
* movlw 0xAA ;W=AAH
* movwf EECON2 ;EECON2 = AAH
* bsf EECON1,WR ;WR=1 (inicia a gravação)
bcf EECON1,WREN ;desabilita a escrita na EEPROM

espera
btfss EECON1,EEIF ;testa flag de término de gravação
goto espera ;EEIF=0 então espera terminar
bcf EECON1,EEIF ;EEIF=1 então zera o flag p/ próximo uso
```

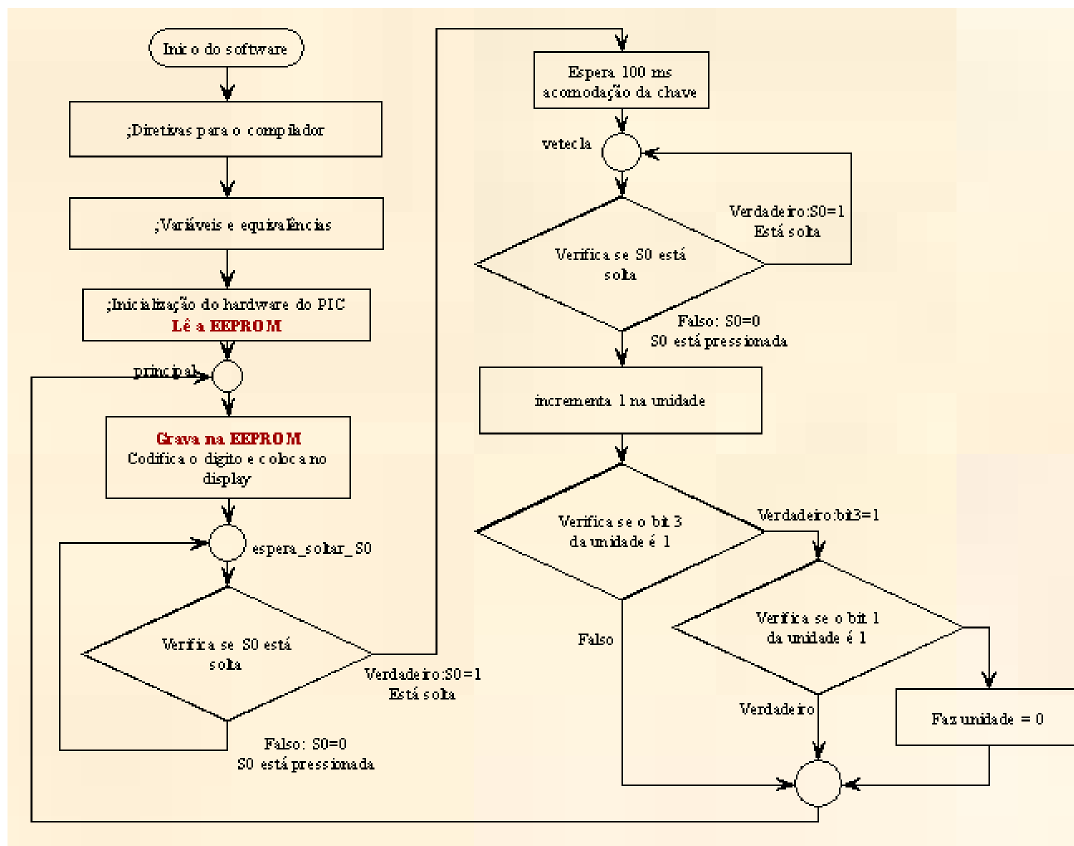
bsf INTCON,GIE ;Habilita as interrupções novamente
Bcf STATUS,RP0 ;volto ao banco 0 fim da gravação na eeprom

As linhas com (*) devem ser colocadas nessa ordem, essa é uma técnica que evita eventuais erros ou desvios inesperados durante a gravação, e associado ao bit WREN permite uma grande margem de segurança nos dados.

Faremos uma aplicação para estudar a gravação na EEPROM. Vamos usar o mesmo circuito da aula 17, aquele com o display de 7 segmentos. A Idéia é o circuito iniciar a contagem, quando ligamos o circuito, com o último número que foi mostrado antes de desligar o circuito.



Veja o fluxograma do código fonte. A leitura da EEPROM se dá no inicio do software, a gravação acontece antes de registrar o digito no display.

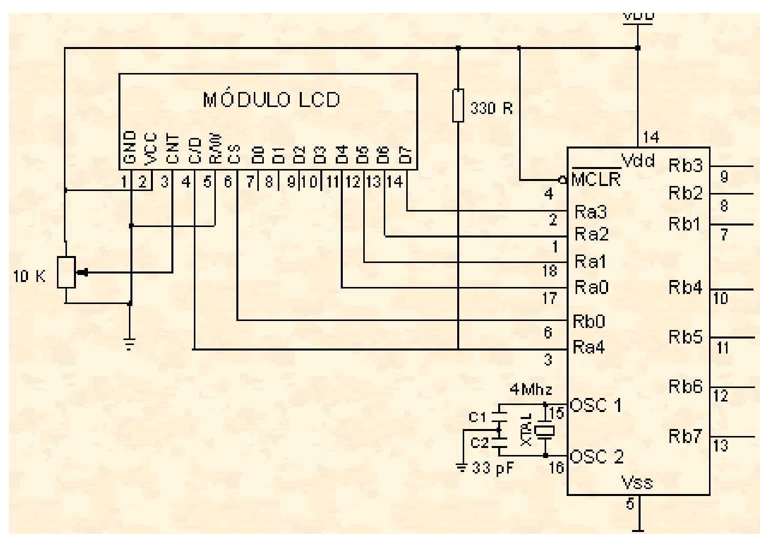


Display LCD inteligente

Uma outra aplicação muito utilizada com microcontrolador é o display de LCD, onde podemos escrever caracteres alfanuméricos segundo código ascii, (ver documentos no CD sobre LCD), esses dispositivos já possuem internamente um microcontrolador dedicado, por isso chamamos de LCD inteligente, e já possui armazenado o código ascii, bastando via software enviar comandos de inicialização, de escrita ou até mesmo de leitura. Existe uma infinidade de LCD no mercado, mas a maioria tem o hardware compatível.

Nosso exemplo de aplicação vai utilizar o modo que utiliza o menor número de pinos possível, usaremos 4 bits para enviar os comandos e os dados. Apesar dos dados serem de 8 bits, esse modo permite que vc divida os 8bits em duas palavras de 4, e o LC

D inteligente monta os 8bits sozinho.



Descrição dos pinos do LCD

Pino	Função	Descrição
1	Alimentação	Terra ou GND
2	Alimentação	VCC ou +5V
3	V0	**Tensão para ajuste de contraste (ver Figura 1)**
4	RS Seleção:	1 - Dado, 0 - Instrução
5	R/W Seleção:	1 - Leitura, 0 - Escrita
6	E Chip select	1 ou (1 → 0) - Habilita, 0 - Desabilitado
7	B0 LSB	Barramento de Dados
8	B1	
9	B2	
10	B3	
11	B4	
12	B5	
13	B6	
14	B7 MSB	
15	A (qdo existir)	Anodo p/ *LED backlight*
16	K (qdo existir)	Catodo p/ *LED backlight*

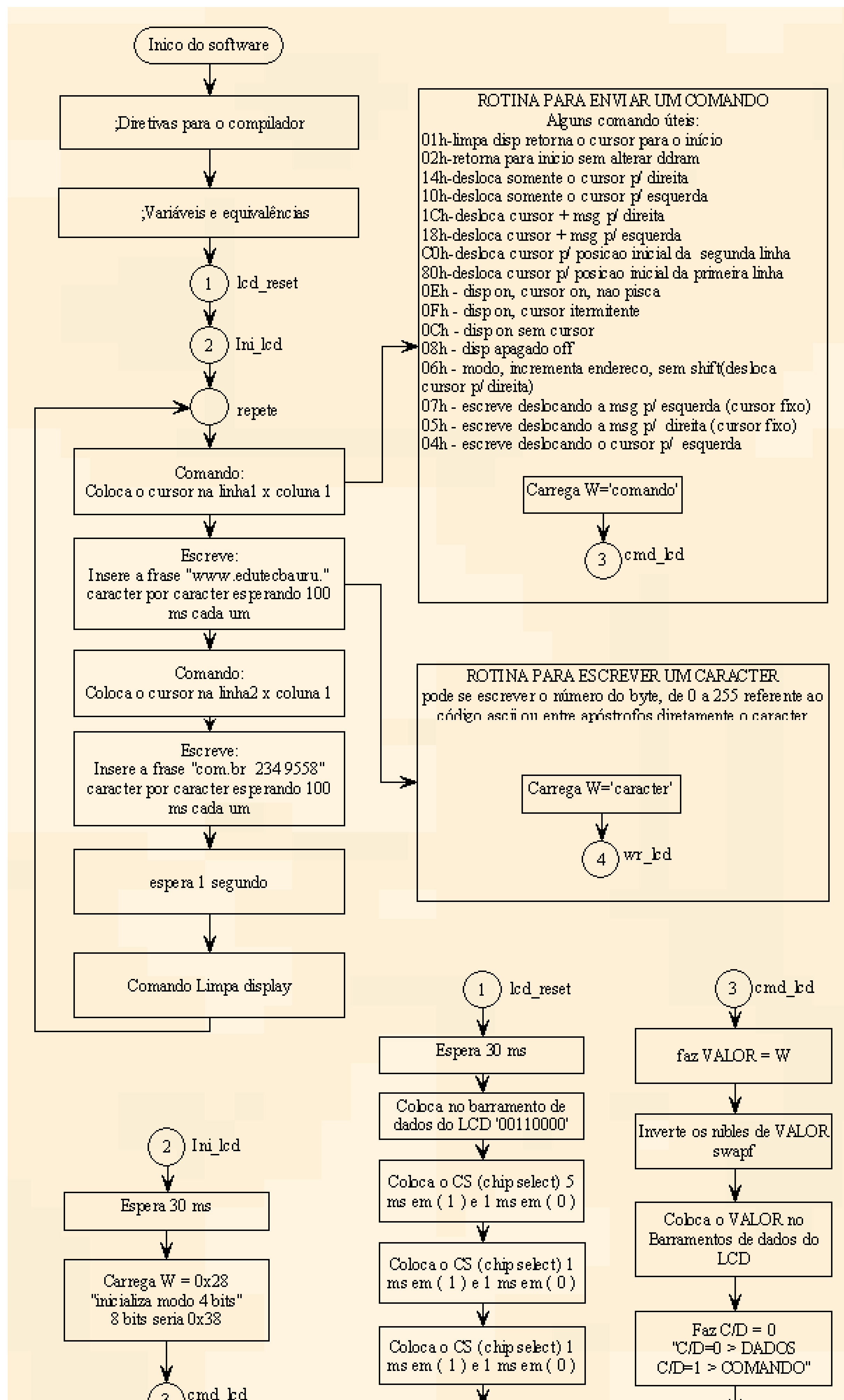
Algumas instruções básicas

DESCRIÇÃO	MODO	RS	R/W	Código (Hexa)
Display	Liga (sem cursor)	0	0	0C
	Desliga	0	0	0A / 08
Limpa Display com Home cursor		0	0	01
Controle do Cursor	Liga	0	0	0E
	Desliga	0	0	0C
	Desloca para Esquerda	0	0	10
	Desloca para Direita	0	0	14
	Cursor Home	0	0	02
	Cursor Piscante	0	0	0D
	Cursor com Alternância	0	0	0F
Sentido de deslocamento do cursor ao entrar com caracter	Para a esquerda	0	0	04
	Para a direita	0	0	06
Deslocamento da mensagem ao entrar com caracter	Para a esquerda	0	0	07
	Para a direita	0	0	05
Deslocamento da mensagem sem entrada de caracter	Para a esquerda	0	0	18
	Para a direita	0	0	1C
End. da primeira posição	primeira linha	0	0	80
	segunda linha	0	0	C0

CODIGO ASCII

HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII
00	0	NULL	20	32	(SP)	40	64	@	60	96	`
01	1	SOH	21	33	!	41	65	A	61	97	a
02	2	STX	22	34	"	42	66	B	62	98	b
HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII
03	3	ETX	23	35	#	43	67	C	63	99	c
04	4	EOT	24	36	$	44	68	D	64	100	d
05	5	ENQ	25	37	%	45	69	E	65	101	e
06	6	ACK	26	38	&	46	70	F	66	102	f
07	7	BEL	27	39	'	47	71	G	67	103	g
08	8	BS	28	40	(	48	72	H	68	104	h
09	9	HT	29	41	)	49	73	I	69	105	i
0A	10	LF	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	\	7C	124	|
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94	^	7E	126	~
1F	31	US	3F	63	?	5F	95	_	7F	127	(sp)

AS ROTINAS DE RESET DO LCD, INICIALIZAÇÃO DO LCD, SÃO RECOMENDAÇÕES DOS FABRICANTES DE LCD


Inico do software
;Diretivas para o compilador
;Variáveis e equivalências
1 lcd_reset
2 Ini_lcd
repete
Comando: Coloca o cursor na linha1 x coluna 1
Escreve: Insere a frase "www.edutecbauru." caracter por caracter esperando 100 ms cada um
Comando: Coloca o cursor na linha2 x coluna 1
Escreve: Insere a frase "com.br 234 9558" caracter por caracter esperando 100 ms cada um
espera 1 segundo
Comando Limpa display
ROTINA PARA ENVIAR UM COMANDO
Alguns comando úteis:
01h-limpa disp retorna o cursor para o início
02h-retorna para inicio sem alterar ddram
14h-desloca somente o cursor p/ direita
10h-desloca somente o cursor p/ esquerda
1Ch-desloca cursor + msg p/ direita
18h-desloca cursor + msg p/ esquerda
C0h-desloca cursor p/ posicao inicial da segunda linha
80h-desloca cursor p/ posicao inicial da primeira linha
0Eh - disp on, cursor on, nao pisca
0Fh - disp on, cursor itermitente
0Ch - disp on sem cursor
08h - disp apagado off
06h - modo, incrementa endereco, sem shift(desloca cursor p/ direita)
07h - escreve deslocando a msg p/ esquerda (cursor fixo)
05h - escreve deslocando a msg p/ direita (cursor fixo)
04h - escreve deslocando o cursor p/ esquerda
Carrega W='comando'
3 cmd_lcd
ROTINA PARA ESCREVER UM CARACTER
pode se escrever o número do byte, de 0 a 255 referente ao código ascii ou entre apóstrofos diretamente o caracter
Carrega W='caracter'
4 wr_lcd
1 lcd_reset
Espera 30 ms
Coloca no barramento de dados do LCD '00110000'
Coloca o CS (chip select) 5 ms em (1) e 1 ms em (0)
Coloca o CS (chip select) 1 ms em (1) e 1 ms em (0)
Coloca o CS (chip select) 1 ms em (1) e 1 ms em (0)
3 cmd_lcd
faz VALOR = W
Inverte os nibles de VALOR swapf
Coloca o VALOR no Barramentos de dados do LCD
Faz C/D = 0
"C/D=0 > DADOS C/D=1 > COMANDO"
2 Ini_lcd
Espera 30 ms
Carrega W = 0x28
"inicializa modo 4 bits"
8 bits seria 0x38
3 cmd_lcd

Monte o circuito do LCD, Faça um novo projeto usando o fonte lcd_01.asm

Considerações Finais

Os microcontroladores da microchip são todos baseados na tecnologia RISC, são projetados para qualquer tipo de aplicação que exigem um alto desempenho e um baixo custo. A Microchip possui mais de 140 tipos de microcontroladores, com uma enorme variedade de configurações de memórias, variedades de periféricos internos. Existem desde microcontroladores simples de 8 pinos até os mais complexos com uma ampla faixa de memória e periféricos. Uma grande vantagem dessa família é que podemos facilmente migrar de uma cpu para outra, facilitando os projetos. Bem como a interligação entre eles também ocorrem de forma tranquila.

Família PIC: Periféricos mais comuns:

Device	Data RAM	ADC	Words	Serial I/O	Speed	Timers	Low Voltage Device
PIC12C508A	25	-	512	-	4	1+WDT	PIC12LC508A
PIC12C509A	41	-	1024	-	4	1+WDT	PIC12LC509A
PIC12CR509A	41	-	1024	-	4	1+WDT	PIC12LCR509A
PIC12CE518	25	-	512	-	4	1+WDT	PIC12LCE518
PIC12CE519	41	-	1024	-	4	1+WDT	PIC12LCE519
PIC12C671	128	4	1024	-	10	1+WDT	PIC12LC671
PIC12C672	128	4	2048	-	10	1+WDT	PIC12LC672
PIC16F630	64	-	1024	-	20	2+WDT	PIC16F630
PIC16F676	64	10	1024	-	20	2+WDT	PIC16F676
PIC12CE673	128	4	1024	-	10	1+WDT	PIC12LCE673
PIC12CE674	128	4	2048	-	10	1+WDT	PIC12LCE674
PIC12F629	64	-	1024	-	20	2+WDT	PIC12F629
PIC12F675	64	4	1024	-	20	2+WDT	PIC12F675
PIC12C508	25	-	512	-	4	1+WDT	-
PIC12C509	41	-	1024	-	4	1+WDT	-
PIC14000	192	8	4096	I²C, SMB	20	2+WDT	-
PIC16C54C	25	-	512	-	40	1+WDT	PIC16LC54C
PIC16CR54C	25	-	512	-	20	1+WDT	PIC16LCR54C
PIC16C55A	24	-	512	-	40	1+WDT	PIC16LC55A
PIC16C56A	25	-	-	-	40	1+WDT	-
PIC16CR56A	25	-	1024	-	20	1+WDT	PIC16LCR56A
PIC16C57C	72	-	2048	-	40	1+WDT	PIC16LC57C
PIC16CR57C	72	5	2048	-	20	1+WDT	PIC16LCR57C
PIC16C58B	73	-	2048	-	40	1+WDT	PIC16LC58B
PIC16CR58B	73	-	2048	-	20	1+WDT	PIC16LCR58B
PIC16C505	72	-	1024	-	20	1+WDT	PIC16LC505
PIC16HV540	25	-	512	-	20	1+WDT	-
PIC16C554	80	-	512	-	20	1+WDT	PIC16LC554
PIC16C558	128	-	2048	-	20	1+WDT	PIC16LC558
PIC16C62B	128	-	2048	I²C, SPI	20	3+WDT	PIC16LC62B
PIC16C63A	192	-	4096	USART, I²C, SPI	20	3+WDT	PIC16LC63A

PIC16CR63	192	-	4096	USART, I²C, SPI	20	3+WDT	PIC16CR63
PIC16C65B	192	-	4096	USART, I²C, SPI	20	3+WDT	PIC16LC65B
PIC16CR65	192	-	4096	USART, I²C, SPI	20	3+WDT	PIC16LCR65
PIC16C66	368	-	8192	USART, I²C, SPI	20	3+WDT	PIC16LC66
PIC16C67	368	-	8192	USART, I²C, SPI	20	3+WDT	PIC16LC67
PIC16C432	128	-	2048	-	20	1+WDT	PIC16LC432
PIC16C433	128	4	2048	-	10	1+WDT	PIC16LC433
PIC16C620A	96	-	512	-	40	1+WDT	PIC16LC620A
PIC16CR620A	96	-	512	-	20	1+ WDT	-
PIC16C621A	96	-	1024	-	40	1+WDT	PIC16LC621A
PIC16C622A	128	-	2048	-	40	1+WDT	PIC16LC622A
PIC16CE623	96	-	512	-	30	1+WDT	PIC16LCE623
PIC16CE624	96	-	1024	-	30	1+WDT	PIC16LCE624
PIC16CE625	128	-	2048	-	30	1+WDT	PIC16LCE625
PIC16C642	176	-	4096	-	20	1+WDT	PIC16LC642
PIC16C662	176	-	4096	-	20	1+WDT	PIC16LC662
PIC16C710	36	4	512	-	20	1+WDT	PIC16LC710
PIC16C711	68	4	1024	-	20	1+WDT	PIC16LC711
PIC16C712	128	4	1024	-	20	3+WDT	PIC16LC712
PIC16C715	128	4	2048	-	20	1+WDT	PIC16LC715
PIC16C716	128	4	2048	-	20	3+WDT	PIC16LC716
PIC16C717	256	6	2048	I²C, SPI	20	3+WDT	PIC16LC717
PIC16C72A	128	5	2048	I²C, SPI	20	3+WDT	PIC16LC72A
PIC16CR72	128	5	2048	I²C, SPI	20	3+WDT	PIC16LCR72
PIC16C73B	192	5	4096	USART, I²C, SPI	20	3+WDT	PIC16LC73B
PIC16C74B	192	8	4096	USART, I²C, SPI	20	3+WDT	PIC16LC74B
PIC16C76	368	5	8192	USART, I²C, SPI	20	3+WDT	PIC16LC76
PIC16C77	368	8	8192	USART, I²C, SPI	20	3+WDT	PIC16LC77
PIC16C770	256	6	2048	I²C, SPI	20	3+WDT	PIC16LC770
PIC16F630	64	-	1024	-	20	2+WDT	PIC16F630
PIC16F676	64	10	1024	-	20	2+WDT	PIC16F676
PIC16C771	256	6	4096	I²C, SPI	20	3+WDT	PIC16LC771
PIC16C773	256	6	4096	USART, I²C, SPI	20	3+WDT	PIC16LC773
PIC16F684	128	8	2048	-	20	3	PIC16F684
PIC16C774	256	10	4096	USART, I²C, SPI	20	3+WDT	PIC16LC774
PIC16C745	256	5	8192	USB,	24	3+WDT	-

				USART			
PIC16C765	256	8	8192	USB, USART	24	3+WDT	-
PIC16C781	128	8	1024	-	20	2+WDT	PIC16LC781
PIC16C782	128	8	2048	-	20	2+WDT	PIC16LC782
PIC16C923	176	-	4096	I²C, SPI	8	3+WDT	PIC16LC923
PIC16C924	176	5	4096	I²C, SPI	8	3+WDT	PIC16LC924
PIC16C925	176	5	4096	SPI, I²C	20	3+WDT	PIC16LC925
PIC16C926	336	5	8192	SPI, I²C	20	3+WDT	PIC16LC926
PIC16F627	224	-	1024	USART	20	3 + WDT	PIC16LF627
PIC16F68	224	-	2048	USART	20	3 + WDT	PIC16LF628
PIC16F72	128	5	2048	I²C, SPI	20	3 + WDT	PIC16LF72
PIC16F73	192	5	4096	I²C, SPI, USART	20	3+WDT	PIC16LF73
PIC16F74	192	8	4096	I²C, SPI, USART	20	3+WDT	PIC16LF74
PIC16F76	368	5	8192	I²C, SPI, USART	20	3+WDT	PIC16LF76
PIC16F77	368	8	8192	I²C, SPI, USART	20	3+WDT	PIC16LF77
PIC16F84A	68	-	1024	-	20	1+WDT	PIC16LF84A
PIC16F870	128	5	2048	USART	20	3+WDT	PIC16LF870
PIC16F871	128	8	2048	USART	20	3+WDT	PIC16LF871
PIC1F872	128	5	2048	I²C, SPI	20	3+WDT	PIC16LF872
PIC16F873	192	5	4096	USART, I²C, SPI	20	3+WDT	PIC16LF873
PIC16F873A	192	5	4096	USART, I²C, SPI	20	3+WDT	PIC16LF873A
PIC16F84	192	8	4096	USART, I²C, SPI	20	3+WDT	PIC16LF874
PIC16F874A	192	8	4096	USART, I²C, SPI	20	3+WDT	PIC16LF874A
PIC16F876	368	8	8192	USART, I²C, SPI	20	3+WDT	PIC16LF876
	368	5	8192	USART, I²C, SPI	20	3+WDT	PIC16LF876A
PIC16F877	368	8	8192	USART, I²C, SPI	20	3+WDT	PIC16LF877
PIC16F877A	368	8	8192	USART, I²C, SPI	20	3+WDT	PIC16LF877A
PIC16C54	25	-	512	-	20	1+WDT	PIC16C54-LP
PIC16CR54A	25	-	512	-	20	1+WDT	PIC16LCR54A
PIC16C54A	25	-	512	-	20	1+WDT	PIC16LC54A, PIC16LV54A
PIC16C55	24	-	512	-	20	1+WDT	PIC16C55-LP
PIC16C56	25	-	1024	-	20	1+WDT	PIC16LC56
PIC16C57	72	-	2048	-	20	1+WDT	PIC16C57-LP
PIC16C62A	128	-	2048	I²C, SPI	20	3+WDT	PIC16LC62A

PIC16C63	192	-	4096	USART, I²C, SPI	20	3+WDT	PIC16LC63
PIC16C64A	128	-	2048	I²C, SPI	20	3+WDT	PIC16LC64A
PIC16C65A	192	-	4096	USART, I²C, SPI	20	3+WDT	PIC16LC65A
PIC16C620	80	-	512	-	20	1+WDT	PIC16LC620
PIC16C621	80	-	1024	-	20	1+WDT	PIC16LC621
PIC16C622	128	-	2048	-	20	1+WDT	PIC16C622
PIC16C71	36	4	1024	-	20	1+WDT	PIC16LC71
PIC16C72	128	5	2048	I²C, SPI	20	3+WDT	PIC16LC72
PIC16C73A	192	5	4096	USART, I²C, SPI	20	3+WDT	PIC16LC73A
PIC16C74A	192	8	4096	USART, I²C, SPI	20	3+WDT	PIC16LC74A
PIC16F83	36	-	512	-	10	1+WDT	PIC16LF83
PIC16CR83	36	-	512	-	10	1+WDT	PIC16LCR83
PIC16F84	68	-	1024	-	10	1+WDT	PIC16LF84
PIC16CR84	68	-	1024	-	10	1+WDT	PIC16LCR84
PIC16F87	368	-	4096	AUSART	20	3	PIC16LF87
PIC16F88	368	7	4096	AUSART	20	3+WDT	PIC16LF88
PIC16F818	128	5	1024	I²C, SPI	20	2/1+WDT	PIC16LF818
PIC16F819	256	5	2048	I²C, SPI	20	2/1+WDT	PIC16LF819
PIC16F737	368	11	4096	MI²C, SPI, AUSART	20	3+WDT	PIC16LF737
PIC16F747	368	14	8192	MI²C, SPI, AUSART	20	3+WDT	PIC16LF747
PIC16F767	368	11	8192	MI²C, SPI, AUSART	20	3+WDT	PIC16LF767
PIC16F777	368	14	8192	MI²C, SPI, AUSART	20	3+WDT	PIC16LF777
PIC16F627A	224	-	1024	USART	20	3+ WDT	PIC16LF627A
PIC16F628A	224	-	2048	USART	20	3+ WDT	PIC16LF628A
PIC16F648A	256	-	4096	USART	20	3+ WDT	PIC16LF648A
PIC17C42A	232	-	2048	USART	33	4+WDT	PIC17LC42A
PIC17C43	454	-	4096	USART	33	4+WDT	PIC17LC43
PIC17CR43	454	-	4096	USART	33	4+WDT	PIC17LCR43
PIC17C44	454	-	8192	USART	33	4+WDT	PIC17LC44
PIC17C752	678	12	8192	USART (2), I²C, SPI	33	4+WDT	PIC17LC752
PIC17C756A	902	12	16384	USART (2), I²C, SPI	33	4+WDT	PIC17LC756A
PIC17C762	678	16	8192	USART (2), I²C, SPI	33	4+WDT	PIC17LC762
PIC17C766	902	16	16384	USART (2), I²C, SPI	33	4+WDT	PIC17LC766
PIC18C242	512	5	8192	AUSART, SPI, I²C	40	4+WDT	PIC18LC242

PIC18C252	1536	5	16384	AUSART, SPI, I²C	40	4+WDT	PIC18LC252
PIC18C442	512	8	8192	AUSART, SPI, I²C	40	4+WDT	PIC18LC442
PIC18C452	1536	8	16384	AUSART, SPI, I²C	40	4+WDT	PIC18LC452
PIC18C601	1536	8	-	AUSART, MI²C, SPI	25	4+WDT	PIC18LC601
PIC18C658	1536	12	16384	AUSART, I²C, SPI, CAN2.0B	40	4+WDT	PIC18LC658
PIC18F6585	3072	12	24576	AUSART, MI²C, SPI, CAN2.0B	40	4+WDT	PIC18LF6585
PIC18C801	1536	12	-	AUSART, MI²C, SPI	25	4+WDT	PIC18LC801
PIC18C858	1536	16	16384	AUSART, I²C, SPI, CAN2.0B	40	4+WDT	PIC18LC858
PIC18F8585	3072	16	24576	AUSART, MI²C, SPI, CAN2.0B	40	4+WDT	PIC18LF8585
PIC18F1220	256	7	2048	EUSART	40	4+WDT	PIC18LF1220
PIC18F1320	256	7	4096	EUSART	40	4+WDT	PIC18LF1320
PIC18F2220	512	10	2048	AUSART, SPI, MI²C	40	4+WDT	PIC18LF2220
PIC18F2320	512	10	4096	AUSART, SPI, MI²C	40	4+WDT	PIC18LF2320
PIC18F242	768	5	8192	AUSART SPI, MI²C	40	4+WDT	PIC18LF242
PIC18F2439	384	-	6144	AUSART MI²C/SPI	40	3+WDT	PIC18LF2439
PIC18F248	768	5	8192	USART, MI²C, SPI, CAN 2.0B	40	4+WDT	PIC18LF248
PIC18F252	1536	5	16384	AUSART SPI, MI²C	40	4+WDT	PIC18LF252
PIC18F2539	1400	5	12288	AUSART MI²C/SPI	40	3+WDT	PIC18LF2539
PIC18F258	1536	5	16384	USART, MI²C, SPI, CAN 2.0B	40	4+WDT	PIC18LF258
PIC18F4220	512	13	2048	AUSART, SPI, MI²C	40	4+WDT	PIC18LF4220
PIC18F4320	512	13	4096	AUSART, SPI, MI²C	40	4+WDT	PIC18LF4320
PIC18F442	768	8	8192	AUSART SPI, MI²C	40	4+WDT	PIC18LF442
PIC18F4439	384	8	6144	AUSART MI²C/SPI	40	3 + WDT	PIC18LF4439
PIC18F448	768	8	8192	USART,	40	4+WDT	PIC18LF448

				MI^{2}C, MSPI, CAN 2.0B			
PIC18F452	1536	8	16384	AUSART SPI, MI^{2}C	40	4+WDT	PIC18LF452
PIC18F4539	1400	8	12288	AUSART MI^{2}C/SPI	40	3 + WDT	PIC18LF4539
PIC18F458	1536	8	16384	USART, MI^{2}C, SPI, CAN 2.0B	40	4+WDT	PIC18LF458
PIC18F6520	2048	12	16384	AUSART (2), SPI, MI^{2}C	40	5+WDT	PIC18LF6520
PIC18F6620	3840	12	32768	AUSART (2), SPI, MI^{2}C	25	5+WDT	PIC18LF6620
PIC18F6680	3072	12	32768	AUSART, MI^{2}C, SPI, CAN2.0B	40	4+WDT	PIC18LF6680
PIC18F6720	3840	12	65536	AUSART (2), SPI, MI^{2}C	25	5+WDT	PIC18LF6720
PIC18F8520	2048	16	16384	AUSART (2), SPI, MI^{2}C	40	5+WDT	PIC18LF8520
PIC18F8620	3840	16	32768	AUSART (2), SPI, MI^{2}C	25	5+WDT	PIC18LF8620
PIC18F8680	3072	16	32768	AUSART, MI^{2}C, SPI, CAN2.0B	40	4+WDT	PIC18LF8680
PIC18F8720	3840	16	65536	AUSART (2), SPI, MI^{2}C	25	5+WDT	PIC18LF8720

Legenda:

- Device: Tipo do PIC temos família 12 - 14 - 16 - 17 - 18
- Data Ram: Tamanho da memória RAM de dados
- ADC: Quantidade de Conversores Analógicos - Digitais
- Words: Tamanho da memória de programa
- Serial I/O : Tipo deEntrada e Saída Serial:
  - I^{2}C, tipo de protocolo de comunicação a dois fios entre chips. protocolo Philips.
  - SMB, Serial compatível com SMBus, Protocolo Intel, é usado para comunicação de baixa velocidade
  - SPI, Serial Peripheral Interface, Modulo de sincronismo para comunicação entre chips, tais como EEPROMS, Shift Register, drives de display, conversores analógicos etc. Compatível com a motorola.
  - USART, Universal Synchronous Assynchronous Receiver Transmitter Periferico para comunicação serial.
  - USB, Periférico de comunicação Serial USB
  - AUSART, Addressable Usart enderessada
  - EUSART, Enhanced Usart." Avançada"
  - CAN2.0B Controller Area Network, Protocolo de comunicação serial para aplicações com alto indice de ruido.
- Speed: Máxima frequência de Clock em MHz
- Timers: Quantidade de Temporizadores e se tem Watch Dog Timer
- Low Voltage Device: Mesmo PIC só que a tensão de alimentação é mais baixa

Além desses periféricos acima, ainda podemos encontrar periféricos cmo EEPROM, PWM, COMPARADORES DE TENSÃO.

Uma boa estudada no site da Microchip www.microchip.com poderá nos dar subsídios para encontrar o melhor microprocessador para nossas aplicações.

Com o nosso curso básico, andamos o primeiro passo para esse mundo das aplicações com microcontroladores

Agora é aplicar tudo que aprendeu, "inventando" pequenas aplicações, montando o circuito, simulando passo a passo o programa, entendo tudo. quando tiver dominando por completo o PIC16F84, pegue um mais avançado, por exemplo o PIC16F627, estude os seus periféricos. procure informações na internet, Faça seus projetos, não tenha medo de errar.